# ODKRYWANIE WŁAŚCIWOŚCI SIECI NEURONOWYCH

## PRZY UŻYCIU PROGRAMÓW W JĘZYKU C#

Opracowanie książki było wspierane
przez firmę **Microsoft Sp. z o.o.**
w ramach programu kooperacji firmy
ze środowiskami akademickimi.

POLSKA AKADEMIA UMIEJĘTNOŚCI
MIĘDZYWYDZIAŁOWA KOMISJA NAUK TECHNICZNYCH

RYSZARD TADEUSIEWICZ
TOMASZ GĄCIARZ, BARBARA BOROWIK, BARTOSZ LEPER

# ODKRYWANIE WŁAŚCIWOŚCI SIECI NEURONOWYCH

## PRZY UŻYCIU PROGRAMÓW W JĘZYKU C#

KRAKÓW 2007

Redaktor Wydawnictwa

Maria Tłustochowska

Redaktor Techniczny

Tomasz Kulawik



Skład główny nakładu:
PAU, ul. Sławkowska 17, 30-016 Kraków
wydawnictwo@pau.krakow.pl

ISBN 978-83-60183-53-3

Polska Akademia Umiejętności
Kraków 2007
Obj.: ark. wyd. 31,45 ; ark. druk. 26,5; nakład 500 egz.
Zakład Graficzny COLONEL
Kraków, ul. Dąbrowskiego 16

# SPIS TREŚCI

## 6. Sieci nieliniowe



## 7. Backpropagation



## 8. Formy uczenia sieci neuronowych



## 9. Sieci neuronowe samouczące się

## 10. Sieci samoorganizujące się



## 11. Sieci rekurencyjne

# PRZEDMOWA

Książka, którą Ci[1] przedstawiam, stanowi trzecią próbę (chcę wierzyć, że najbardziej udaną) opracowania zasygnalizowanego w tytule tematu.

Pierwszą taką próbą, dość dobrze przyjętą przez Czytelników, była seria artykułów popularyzujących problematykę sieci neuronowych na łamach popularnego miesięcznika informatycznego ***Enter*** (wychodzącego w Polsce, ale czytanego chętnie także w kilku sąsiednich krajach, na przykład na Słowacji). Doszło do tego w taki sposób, że pod koniec 1994 roku redakcja tego miesięcznika zwróciła się do mnie z prośbą, żebym spróbował napisać artykuł wyjaśniający czytelnikom tego miesięcznika maksymalnie popularnie, o co chodzi z tymi **sieciami neuronowymi**, o których tyle się pisze i mówi? W odpowiedzi zamiast jednego artykułu – napisałem cały ich cykl, który ukazywał się we wszystkich kolejnych numerach czasopisma ***Enter*** od stycznia 1995 roku do marca 1996 roku włącznie. Artykuły te, zamieszczone (z moją chętną aprobatą!) w kilku portalach internetowych[2], bywają

[1] Książka ta jest kierowana głównie do **młodych** Czytelników, to znaczy uczniów szkół średnich i studentów. Z tego względu przy jej pisaniu przyjęto i konsekwentnie stosowano w całym tekście zasadę zwracania się do Osoby Czytającej w sposób przyjaźnie bezpośredni, z użyciem formy „***Ty***".

Jeśli książkę będą czytały także Osoby starsze i dostojne (co sprawi autorom wielką przyjemność!), to usilnie prosimy, by nie odbierały one tej bezpośredniej formy jako objawu poufałości czy też (nie daj Boże!) lekceważenia Czytelnika. Zapewniamy, że darzymy wszystkich Czytelników ogromnym szacunkiem i sympatią, ale przekonaliśmy się, że pewne stwierdzenia można wypowiedzieć **jaśniej** i nadać im bardziej przekonującą formę przy bezpośrednim adresowaniu ich do Odbiorcy, więc taką właśnie formułę komunikacji wybraliśmy i będziemy jej używali. Dla symetrii również narrator książki będzie wypowiadał się zawsze w pierwszej osobie liczby pojedynczej, chociaż książka ma wielu autorów, bo chcemy jak najbardziej „skrócić dystans" dzielący autora i Czytelnika. Mamy nadzieję, że to nikogo nie urazi, nie zrazi, ani (tym bardziej) nie obrazi!

[2] Patrz na przykład: http://www.jasinski.us/index.php?n=sieci_neuronowe

czytane do dnia dzisiejszego, o czym świadczą liczne listy nadchodzące od Czytelników.

Widząc powodzenie tych artykułów redaktor naczelny Akademickiej Oficyny Wydawniczej, prof. Leonard Bolc, zaproponował, żeby artykuły te zebrać i wydać w formie książkowej[3]. Opracowanie i wydanie tej książki trochę trwało, bo materiał dla zwartego woluminu musiał być bardziej zintegrowany i uporządkowany, niż to było konieczne podczas pisania z miesiąca na miesiąc kolejnych odcinków cyklu artykułów, ale na koniec ukazała się książka: **Tadeusiewicz R.:** ***Elementarne wprowadzenie do sieci neuronowych z przykładowymi programami*****, Akademicka Oficyna Wydawnicza, Warszawa 1998**, którą można wskazać jako bezpośrednią poprzedniczkę tej książki, oddawanej w tym momencie do rąk polskiego czytelnika. Książka ta bardzo szybko zniknęła z półek księgarskich, a jej egzemplarze dostępne w wielu bibliotekach zostały „zaczytane" praktycznie do granic fizycznego unicestwienia – co stało się bezpośrednią przyczyną podjęcia próby opracowania kolejnego wydania tej książki. Okazało się jednak, że postęp w dziedzinie sieci neuronowych był tak szybki, że już po kilku latach od pierwszej publikacji książka *Elementarne wprowadzenie…* stała się tak bardzo przestarzała, że zamiast kolejnego wydania (poprawionego i uzupełnionego) zdecydowałem się raczej na przedstawienie Czytelnikom **zupełnie nowej książki** – tej właśnie, którą trzymasz w ręce.

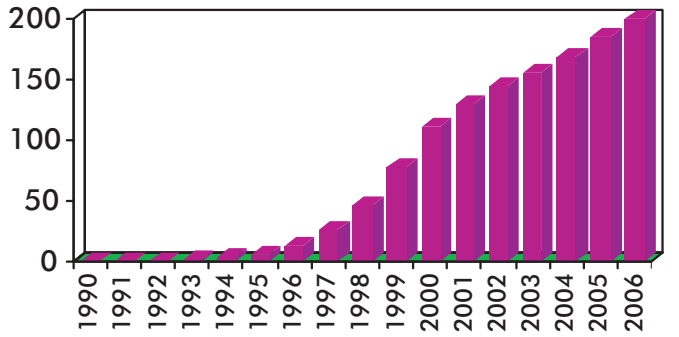
Jakie jest uzasadnienie takiej decyzji?

Żeby to dokładniej wyjaśnić, przyjrzyjmy się motywacjom, jakie towarzyszyły przygotowaniu pierwszych artykułów do miesięcznika ***Enter***, oraz motywacjom, które towarzyszą nowemu wydaniu książki na ten sam temat tu i teraz.

Artykuły (i książka będąca potem ich kompilacją) powstały na prośbę odpowiednich redaktorów – odpowiednio miesięcznika oraz oficyny wydawniczej. Prośby te osadzone były w specyficznym kontekście tamtych lat, który warto tu w skrócie przypomnieć, bo to ma związek zarówno z formą, jak i z merytoryczną zawartością tej książki. Otóż właśnie w początkowych latach 90. miało miejsce zjawisko, które nawet w dynamicznie zmieniającym się świecie informatyki zdarza się zdecydowanie rzadko. Zapanowała wtedy moda (by nie powiedzieć – epidemia!) stosowania do wszystkiego tak zwa-

[3] Warto w tym miejscu nadmienić, że istnieje długa i dobra tradycja wydawania w formie książkowej dzieł, które wcześniej funkcjonowały jako seria odcinków publikowanych w prasie. *Toute proportion gardée*, czyli zachowując wszystkie właściwe proporcje, można nadmienić, że dokładnie sto lat wcześniej, w latach 1895–1896, na łamach „Gazety Polskiej" ukazywała się w odcinkach powieść *Ouo vadis*, za którą Henryk Sienkiewicz otrzymał potem Nagrodę Nobla!

nych „neurokomputerów[4]". Moda ta trwa zresztą do dzisiaj i to właśnie ona jest zapewne uzasadnieniem faktu, że czytasz tę książkę. Co prawda, trochę się od czasu lat 90. zmieniło: po pewnym czasie (gdy już wygasła pierwsza fala bezkrytycznego entuzjazmu) futurystyczną nazwę ***neurokomputer*** porzucono i odpowiednie narzędzia informatyczne zaczęto nazywać rozsądniej i spokojniej po prostu **sieciami neuronowymi**, ale moda na ich stosowanie trwała nadal, a nawet się nasilała, co można prześledzić na rysunku P.1.




Rys. P.1. Wielkość zysków (w mln USD) pochodząca ze sprzedaży oprogramowania dla tworzenia i wykorzystywania sieci neuronowych. Dane dla rynku amerykańskiego. Dla 2006 roku dane estymowane

Powód opisanego zjawiska był prosty: specjaliści różnych dziedzin w latach 90. „odkryli", że sieci neuronowe mogą stanowić niezwykle skuteczne narzędzie do informatycznego rozwiązywania ich problemów. Napisałem „odkryli" w cudzysłowie, ponieważ te „cudowne" właściwości nowego narzędzia, które entuzjastycznie opisywano i dyskutowano w fachowym informatycznym piśmiennictwie tamtych lat, były już wcześniej dobrze znane, tyle tylko że niezbyt licznym specjalistom: biocybernetykom, a dokładniej mówiąc – neurocybernetykom.

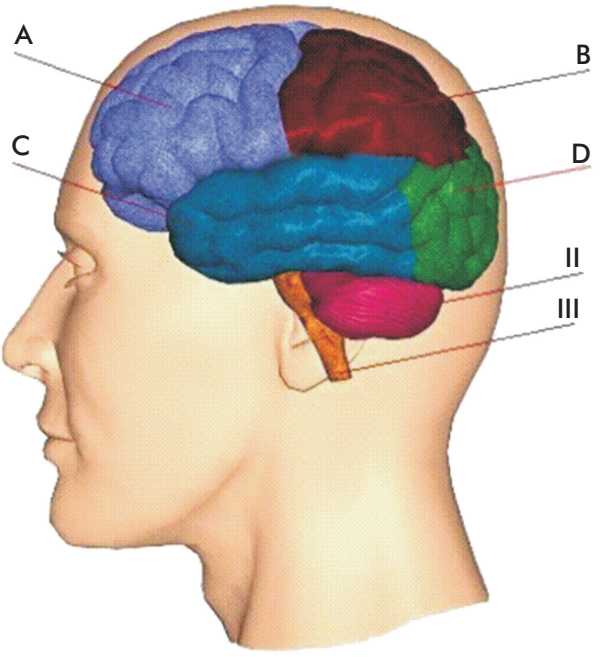
W Polsce ja byłem wtedy takim unikatowym specjalistą, ponieważ od co najmniej 20 lat zajmowałem się modelowaniem komputerowym ludzkiego mózgu[5], a w dodatku chętnie i często popularyzowałem wiedzę na ten temat[6].

[4] Patrz na przykład: Tadeusiewicz R.: *Neurokomputery – wczoraj, dziś, jutro*, „Wiedza i Życie", nr 7, 1990, s. 19–26.

[5] Jak się wydaje, pierwszą publikacją na ten temat była praca: Tadeusiewicz R.: *Wybrane zagadnienia cyfrowego modelowania fragmentów systemu nerwowego. W materiałach II Ogólnopolskiego Sympozjum: System – Modelowanie – Sterowanie*, Zakopane 1974, s. 112–114.

[6] Jak się wydaje jednymi z pierwszych popularnych opracowań, dotyczących rozważanej tu tematyki, były artykuły: Tadeusiewicz R.: *Zadziwiające własności komórki nerwowej*, „Młody Technik", nr 6, 1977, s. 5–12 oraz Tadeusiewicz R.: *Co to jest perceptron?*, „Młody Technik", nr 4, 1979, s. 36–42, a znaczący wpływ na dość powszechny wzrost zainteresowa-

Ponadto na początku lat 90. napisałem pierwszą w Polsce książkę na temat sieci neuronowych, która do dzisiaj jest czytana i cytowana, a obecnie można ją studiować na całym świecie, ponieważ jest udostępniana w zbiorach Polskiej Biblioteki Internetowej pod adresem **www.pbi.edu.pl** albo w zbiorach Akademickiej Biblioteki Cyfrowej pod adresem **http://abc.agh.edu.pl/** . Ponieważ książka jest często czytana, więc możliwy jest także bezpośredni natychmiastowy dostęp do jej treści poprzez jej własny adres internetowy **http://winntbg.bg.agh.edu.pl/skrypty/0001/** .




Rys. P.2. Sieci neuronowe stanowią informatyczne naśladownictwo niektórych właściwości wykrytych podczas badania mózgu ludzi i zwierząt

Jednak wspomniana książka była i jest przeznaczona dla profesjonalistów, natomiast w latach 90. było wiele osób po prostu **zaciekawionych** sieciami neuronowymi, ale nie mających profesjonalnego przygotowania w tym zakresie, a tymczasem dla nich nie było żadnej stosownej literatury.

---

nia tymi problemami miały prace: Tadeusiewicz R.: *Cybernetyczny model komórki nerwowej*, „Problemy", nr 9, 1983, s. 11–18 oraz Tadeusiewicz R.: *Sieci neuropodobne*, „Problemy", nr 5, 1985, s. 2–10.

Nie ukrywam, że pochlebił mi bardzo fakt, że to właśnie mnie poproszono o wyjaśnienie szerokiemu ogółowi, czym są sieci neuronowe, więc podjąłem się tego dzieła. Traktowałem to początkowo jako zadanie na jeden wieczór, bo pozornie cóż może być prostszego od popularnego wytłumaczenia tego, co się dobrze zna i bardzo lubi? Tymczasem okazało się, że tego, co chciałem opowiedzieć o sieciach neuronowych, jest więcej, niż się mogłem spodziewać. W rezultacie popularyzowanie sieci neuronowych stało się dla mnie pasjonującym, ale i bardzo czasochłonnym zajęciem na kilka kolejnych lat, bo sam nie zdawałem sobie sprawy, jak wiele fascynujących rzeczy można o tych sieciach powiedzieć (i pokazać praktycznie!) zaciekawionym Czytelnikom. W związku z tym temat rozrastał mi się pod piórem, zajmował kolejne numery miesięcznika (w sumie moje artykuły na ten temat ukazywały się nieprzerwanie przez ponad rok!), a potem ten sam materiał zebrany i uporządkowany stał się podstawą do napisania poprzedniej wersji książki.

Początkowo zamierzałem wyjaśnić Czytelnikom głównie to, że w fascynujących ludzi właściwościach sieci neuronowych odbijają się tylko niektóre (i to bardzo skromne) fragmenty tej ogromnej i bardzo ciekawej wiedzy, jaką przez wiele dziesięcioleci gromadzili badacze, śledzący procesy przetwarzania informacji, uczenia się oraz formowania się inteligentnych zachowań w zakamarkach mózgu ludzi i zwierząt (rys. P.2).

Prace tych badaczy, zapoczątkowane przez rosyjskiego uczonego Pawłowa, zaowocowały całą serią odkryć uhonorowanych między innymi najcenniejszym trofeum, jakie uczony może uzyskać: nagrodą Nobla. W tabeli P.1 pokazuję przykładowe zestawienie tych laureatów nagród Nobla, którzy przyczynili się do zrozumienia zasad, według których funkcjonuje biologiczny mózg, a przez to stworzyli przesłanki do zbudowania technicznych sieci neuronowych.

Jak widać z tabeli, biologowie na początku lat 90. wiedzieli już bardzo dużo o mózgu, a biocybernetycy od lat budowali modele cybernetyczne (matematyczne oraz symulacyjne, komputerowe) wszystkich tych mechanizmów, które wykrywano w mózgu. Przy okazji stwierdzono ponad wszelką wątpliwość, że mózg gromadzi i przetwarza informacje sprawniej i mądrzej niż systemy komputerowe, chociaż te ostatnie górują nad nim szybkością działania i pojemnością pamięci masowych.

Po stwierdzeniu, że cybernetyczny model mózgu lepiej kojarzy informacje niż program komputerowy, sprawniej klasyfikuje i rozpoznaje złożone zestawy danych (na przykład obrazy), inteligentniej wyszukuje dane na podstawie ich treści i zachodzących między nimi asocjacji itd. – było oczywiste, że oparcie na podobnych mechanizmach funkcjonowania użytecznych narzędzi informatycznych jest tylko kwestią czasu. W latach 90. nowością było

**Tabela P.1: Nagrody Nobla** związane z tymi badaniami układu nerwowego, których wyniki wykorzystano pośrednio lub bezpośrednio w sieciach neuronowych

| | | |
|---|---|---|
| 1904 | Pavlov I.P. | teoria odruchów warunkowych |
| 1906 | Golgi C. | badanie struktury układu nerwowego |
| 1906 | Ramón y Cajal S. | odkrycie, że mózg składa się z sieci oddzielnych neuronów |
| 1920 | Krogh S.A. | opisanie funkcji regulacyjnych w organizmie |
| 1932 | Sherrington Ch. S. | badania sterowania nerwowego pracy mięśni |
| 1936 | Dale H., Hallett L.O. | odkrycie chemicznej transmisji impulsów nerwowych |
| 1944 | Erlanger J., Gasser H. S. | procesy w pojedynczym włóknie nerwowym |
| 1949 | Hess W.R. | odkrycie funkcji śródmózgowia |
| 1963 | Eccles J.C., Hodgkin A.L., Huxley A.F. | mechanizm elektrycznej aktywności neuronu |
| 1969 | Granit R., Hartline H.K., Wald G. | fizjologia widzenia |
| 1970 | Katz B., Von Euler U., Axelrod J. | transmisja informacji humoralnej w komórkach nerwowych |
| 1974 | Claude A., De Duve Ch., Palade G. | badania strukturalnej i funkcjonalnej organizacji komórki |
| 1977 | Guillemin R., Schally A., Yalow R. | badania hormonów mózgu |
| 1981 | Sperry R. | odkrycia dotyczące funkcjonalnej specjalizacji półkul móżdżku |
| 1981 | Hubel D.H., Wiesel T. | odkrycie zasad przetwarzania informacji w systemie wzrokowym |
| 1991 | Neher E., Sakmann B. | funkcje kanałów jonowych w komórkach nerwowych |

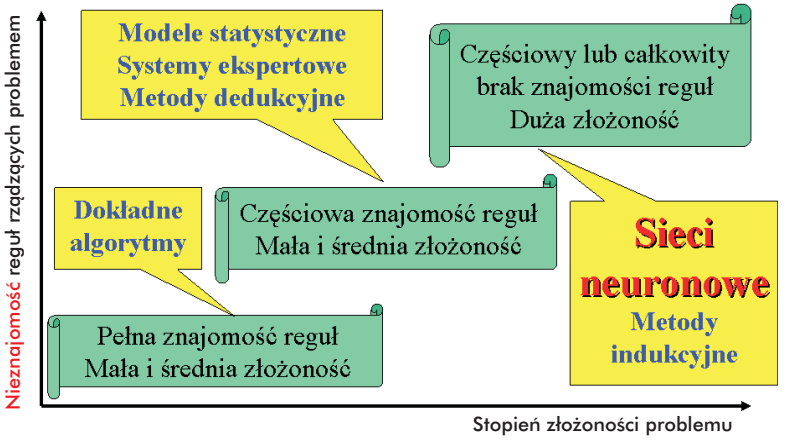
głównie to, że możliwości te poznali inżynierowie, badacze i praktycy, którzy wcześniej ich nie znali, chociaż naukowcy zajmujący się neurofizjologią oraz kognitywistyką pisali o nich od lat. Dlatego sensacyjny ton wielu doniesień o sukcesach (znaczących!) uzyskiwanych dzięki sieciom neuronowym był przez specjalistów biocybernetyków przyjmowany z pewną dozą pobłażliwej wyrozumiałości. Jednak ponieważ zjawisko to przybrało wtedy skalę prawdziwie masową, a ponadto było godne uwagi i budziło szerokie zainteresowanie (między innymi czytelników wspomnianego miesięcznika ***Enter***), przeto pojawiła się prośba, żebym także i o tym coś napisał – i ja to zrobiłem.

Pozostawmy jednak na uboczu ten wątek historyczny i przejdźmy do omówienia genezy i zawartości tej książki, którą w tej chwili trzymasz w ręce.

Z perspektywy pierwszej dekady XXI wieku można stwierdzić, że moda na stosowanie sieci neuronowych, jaka zaczęła narastać w latach 90., trwa do dzisiaj. Nie wynika to tylko z powszechnie znanej i stale obserwowanej skłonności informatyków do fascynowania się każdą kolejną nowością, jaką

przynosi postęp techniki oraz rozwój cywilizacji. Zachwyt nad sieciami neuronowymi miał i ma także swoje dodatkowe, konkretne i ważne przyczyny. Główną przesłanką dla ogromnej popularności sieci były i są **naprawdę doskonałe wyniki**, uzyskiwane przy pomocy tego nowego narzędzia podczas rozwiązywania wielu problemów, o których od dawna było wiadomo, że są szczególnie trudne. Sensacyjne doniesienia pionierów, którzy jako pierwsi wypróbowali sieci neuronowe i opisali ich znakomite właściwości w swoich książkach i publikacjach, zachęciły licznych naśladowców. W ciągu całej dekady lat 90. i w pierwszych latach XXI stulecia doniesienia na temat tego, kto i do czego te sieci z sukcesem zastosował, pojawiały się jak przysłowiowe „grzyby po deszczu", dlatego pisząc jeden z moich kolejnych artykułów na ten temat, użyłem nawet kiedyś żartobliwego określenia, że „*obecnie nieznajomość sieci neuronowych w niektórych sferach zaczęła być traktowana wręcz jako nietakt towarzyski!*"

Abyś tych zapewnień nie traktował jako gołosłownych ogólników, spróbuję teraz pokazać Ci, gdzie i do czego sieci neuronowe się stosuje, a z tego wyniknie także zbiorczy pogląd na temat tego, dlaczego mają one tak ciekawe właściwości. Popatrz na rysunek P.3. Na rysunku tym pokazałem (w dosyć umowny sposób) klasyfikację zadań, jakie są wykonywane przez różne systemy informatyczne. Jest chyba dla Ciebie oczywiste, że są wśród nich zadania łatwiejsze i trudniejsze, zatem na osi poziomej pokazanego wykresu zaznaczyłem jakąś miarę stopnia **trudności** zadania. Prawdę powiedziawszy dokładne wyskalowanie tej osi nie byłoby łatwe, bo nie do końca wiadomo, w jaki sposób należy mierzyć tę trudność. Ale z pewnością można wyróżnić zadania łatwe (bliżej lewej strony wykresu) oraz zadania trudne (po prawej stronie) – i to nam na razie wystarczy.




Rys. P.3. Charakterystyka zadań informatycznych o różnym stopniu trudności oraz lokalizacja tych zadań, do realizacji których szczególnie dobrze nadają się sieci neuronowe

Trudność rozwiązywanego zadania nie jest jedyną miarą kłopotów, z jakimi musi się zmierzyć informatyk, który chce je rozwiązać. Drugim wymiarem problemu jest bowiem **dostępność wiedzy**, na której można się oprzeć budując rozwiązanie. Dla niektórych zadań reguły są dokładnie określone, chociaż liczba tych reguł albo stopień ich komplikacji mogą powodować, że rozwiązywane zadanie będzie mimo to trudne. Na przykład twórca oprogramowania dla dużego banku musi się sporo napracować, chociaż reguły są tu dobrze znane i w miarę proste: gdy klient wpłaca pieniądze, to stan jego konta należy powiększyć, a gdy je wypłaca, to trzeba saldo pomniejszyć. Ale tych klientów jest dużo, transakcji jeszcze więcej – i to powoduje, że program dla banku jest skomplikowany i drogi. Inny rodzaj trudności ma fizyk rozwiązujący modele matematyczne opisujące wnętrze atomu. Równań jest tu mało, ale są one bardzo trudne do rozwiązania. W obu przywołanych przykładach miara komplikacji, jaką jest **nieznajomość reguł**, jest jednak zerowa, bo same reguły **są** znane i mogą być wykorzystane przy budowie narzędzia rozwiązującego zadanie.

Wiemy, jak używać komputerów przy rozwiązywaniu zadań cechujących się pełną znajomością reguł: należy na podstawie reguł zbudować algorytm, a na podstawie algorytmu napisać odpowiedni program. Nie twierdzę, że takie napisanie programu jest zawsze proste i łatwe, ale przynajmniej dokładnie wiadomo, **co** należy zrobić oraz **jak** to należy zrobić.

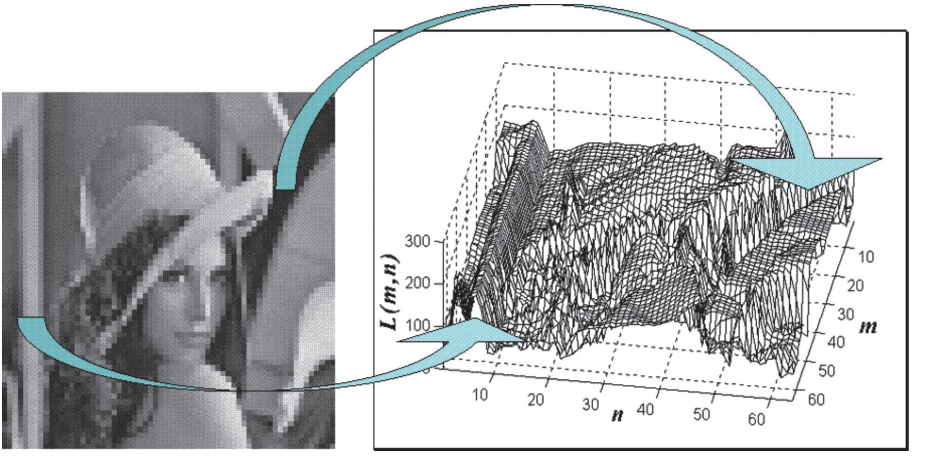
Sytuacja może jednak być bardziej skomplikowana. Często musimy rozwiązywać problemy, w których reguły **nie są znane**. Jeśli chcemy rozwiązać te zadania, to możemy to czasem uczynić opierając się na spostrzeżeniu, że zjawiska albo procesy, których one dotyczą, są w jakimś sensie powtarzalne. Mamy wtedy do czynienia z sytuacją symbolicznie zlokalizowaną w centralnej części rysunku P.3. Znajomość reguł nie jest wtedy kompletna, a znane reguły nie są do końca pewne – ale mimo tych braków możemy się posłużyć wygodną techniką **dedukcji:** Próbujemy wytworzyć ogólną regułę (lub częściej zbiór ogólnych reguł), z których potem będziemy korzystać przy rozwiązywaniu każdego konkretnego problemu szczegółowego.

Programy budowane dla tej klasy zagadnień uwzględniają zwykle fakt, że nasza wiedza jest niepewna i niepełna, dlatego używamy wtedy chętnie narzędzi statystycznych (na przykład w ekonomii lub agrotechnice), a także opieramy się na wiedzy niealgorytmizowanej. Przykładowo przy rozwiązywaniu takich zadań często opieramy się na zdroworozsądkowej wiedzy ekspertów. Wiedza taka, oparta na latach doświadczeń, może być zaskakująco skuteczna, co na przykład wykorzystuje się w popularnych systemach ekspertowych sugerujących diagnozy lub doradzających najwłaściwsze metody terapii w medycynie.

Istnieje jednak jeszcze bardziej złożona sytuacja, w której niepodobna podać **żadnych** reguł. W takich problemach jedyne, czym dysponujemy, to szereg przykładów zadań, które zostały poprawnie rozwiązane. Te poprawne rozwiązania mogą pochodzić z obserwacji zachowania systemu, którego właściwości chcemy modelować, chociaż nie wiemy, jaka jest jego wewnętrzna budowa i dlaczego działa tak, jak to widzimy. Czasami wzorcowe rozwiązania, które będziemy chcieli naśladować, podawane są przez kogoś, kto umie to robić, ale sam nie wie, jak osiąga wymagany rezultat, bo odpowiednie procesy analizy sytuacji i podejmowania decyzji toczą się w sferze nie podlegającej kontroli ze strony świadomości.

W pierwszej chwili być może sądzisz Czytelniku, że takich sytuacji nie ma wcale albo jest ich niewiele. Mylisz się! Nasz mózg rozwiązuje takie zadania bez przerwy, na przykład w związku z zadaniami określanymi w psychologii mianem **percepcji**. Subiektywnie sprawa jest bardzo prosta: Oto idziesz ulicą i spotykasz koleżankę. Rozpoznajesz ją bez trudu – ta twarz, ta sylwetka – nie ma wątpliwości, kto to taki. Ale wyobraź sobie, że tę samą czynność powinien wykonać komputer, na przykład po to, żeby jednych ludzi wpuszczać do Twojego domu, a innych nie. Początek jest zachęcająco łatwy, bo powszechnie dostępne obecnie cyfrowe aparaty fotograficzne oraz kamery video wytwarzają obraz w takiej formie, że może on bez trudu zostać wprowadzony do komputera (patrz rys. P.4). Pojawia się jednak pytanie, co dalej?

Obraz wprowadzony do pamięci komputera jest rozbity na miliony punktów (pikseli), a jasność lub barwa każdego z tych punktów jest zakodowana w




Rys. P.4. Cyfrowy obraz wprowadzony do pamięci komputera ma wygodną obliczeniowo postać dwuwymiarowej funkcji i łatwo można odnaleźć charakterystyczne elementy obrazu w postaci szczegółów tej funkcji. Trudno jest jednak podać algorytm rozpoznawania osoby przedstawionej na obrazie na podstawie wartości tej funkcji

formie cyfrowej, co powoduje, że cyfrowy obraz może być łatwo obrabiany przez komputer, który jest wszak maszyną obliczającą. Co więcej, można się łatwo przekonać, że w tych milionach wartości numerycznych odpowiadających poszczególnym pikselom **jest** zawarta informacja potrzebna do tego, żeby rozpoznać osobę przedstawioną na obrazie, wystarczy bowiem obraz wyświetlić na ekranie albo wydrukować na drukarce, żeby stwierdzić ponad wszelką wątpliwość: „*To jest Lena*".

No dobrze, ale jak podać algorytm, który te miliony wartości opisujących szarość poszczególnych punktów obrazu, zamieni na takie samo stwierdzenie, ale wygenerowane automatycznie przez komputer? Co więcej, algorytm ten musi potrafić rozpoznać znajomą twarz niezależnie od tego, czy jest fotografowana *en face* czy z profilu, czy jest bliżej czy dalej, czy jest lepiej czy gorzej oświetlona itd. Trzeba mieć sposób na to, żeby nie przeszkadzały nam w skutecznym rozpoznaniu danej osoby zmienne elementy jej mimiki (na przykład obecność lub brak uśmiechu), zmienne nakrycia głowy, różne tło, na jakim ją obserwujemy itd. – patrz rys P.5. Dla naszego mózgu sprawy te nie stanowią żadnego problemu, ale gdy chcemy tę naszą umiejętność zaprogramować w postaci ogólnego algorytmu dla komputera, to okazuje się, że nie potrafimy tego zrobić!

W podobnych zadaniach przy przekazywaniu określonej umiejętności innym ludziom posługujemy się **prezentacją przykładów**. Na przykład pokazujemy dziecku obrazki i mówimy: *To jest litera A, to także jest litera A, chociaż jest mniejsza, a tutaj także masz literę A, która wygląda nieco inaczej, bo jest wydrukowana inną czcionką – ale czyta się tak samo.* Postępując tak liczymy na to (na ogół słusznie), że nauczany odbiorca wiadomości potrafi


Rys. P.5. Ta sama twarz może różnie wyglądać na obrazie zależnie od oświetlenia (a), zależnie od położenia (b) oraz zależnie od mimiki (c). Utrudnia to bardzo zadanie automatycznego rozpoznawania!

z tych wielu szczegółowych pokazów sam sobie wytworzyć jakiś syntetyczny sposób rozumowania, który pozwoli mu skutecznie rozpoznawać wszystkie pokazane przykładowe formy rozważanych obrazów, ale także i inne obrazy tej samej litery (albo tej samej osoby czy innego rozpoznawanego obiektu). I co więcej, przy nauczaniu ludzi lub przy tresurze zwierząt uzyskujemy bardzo dobre rezultaty przy uogólnianiu wiedzy uzyskanej poprzez prezentację konkretnych przykładów, gdyż uczone dziecko albo wytresowany pies potrafią rozwiązać także zadania, których bezpośrednio im wcześniej nie pokazywano – pod warunkiem, że będą on wystarczająco podobne do zaprezentowanych podczas nauki przykładów.

Wymagane w tym przypadku podejście do rozwiązywania problemów nazywa się **metodą indukcyjną**. Polega ona na tym, że w odróżnieniu od wcześniej omawianej dedukcji – w tym przypadku **nie znamy żadnej ogólnej reguły,** na podstawie której moglibyśmy rozwiązywać konkretne szczegółowe problemy. Zamiast tego mamy jednak garść przykładów konkretnych zadań, dla których znane są poprawne rozwiązania. Umysł człowieka zdolny jest do tego, że potrafi takie konkretne przykłady analizować, wyciągać z nich wnioski, a następnie te wnioski tak uogólniać. Również mózg tresowanego zwierzęcia, na przykład szczura w labiryncie, potrafi tak się nauczyć reguł znajdowania właściwej drogi, że w efekcie udaje się rozwiązać nie tylko te zagadnienia, których używano wcześniej jako przykładów, ale także inne, wykazujące określony stopień podobieństwa do tych zadań, których używano w trakcie procesu uczenia.

Jak wspomniałem, specjaliści zajmujący się neurocybernetyką znali już wcześniej ten ogromny potencjał, jaki od **zawsze** tkwił w poznawanych przez biologów umiejętnościach uczenia się mózgu (ludzi i zwierząt) oraz w neuronowych metodach przetwarzania informacji. Jednak większość pozostałych naukowców i praktyków, zwłaszcza tych od lat korzystających z komputera, uważało, że znajomość działania precyzyjnego algorytmu albo przynajmniej dostatecznie silnej reguły dedukcyjnej jest **niezbędnym** warunkiem uzyskania skutecznego rozwiązania każdego problemu teoretycznego lub aplikacyjnego. Gdy się okazało, że sieci neuronowe potrafią rozwiązywać różne problemy bez programowania, tylko metodą wnioskowania przez analogię, fakt ten powitano z wielkim zaciekawieniem. Gdy potem dodatkowo okazało się, że mogą one same odkrywać nieznane wcześniej reguły postępowania oraz rozwiązywać takie zadania, dla których nikt z ludzi nie potrafiłby podać sposobu ich rozwiązywania – zapanował ogromny entuzjazm.

Doniesienia naukowe strzelały jak fajerwerki, a formułowane w nich wnioski wskazywały często na bezkrytyczną fascynację ich autorów tym nowym narzędziem. Ta fala skutecznych i efektownych sukcesów sieci neuronowych

rozbudziła ponownie nadzieje na to, że oto odkryty został nowy „kamień filozoficzny", bezskutecznie poszukiwany przez wiele stuleci przez alchemików i astrologów. Oczywiście to ostatnie zdanie trzeba traktować jako przenośnię, bowiem w sieciach neuronowych nie upatrywano nigdy narzędzia do zamiany ołowiu w złoto, ale równie naiwnie sądzono, że z pomocą tej nowej techniki uda się rozwiązać wszystko i zawsze.

Neurocybernetycy znali już wcześniej zdolności uczenia się i uogólniania wiedzy, charakterystyczne dla systemów modelujących strukturę i zachowanie ludzkiego mózgu, ale byli także świadomi, że możliwości, jakie tkwią w tej nowej technice, są ograniczone. Dlatego odnosili się do tych doniesień naukowych z zainteresowaniem, bo ludzka pomysłowość w stosowaniu sieci neuronowych okazała się niewiarygodnie duża, ale także i ze spokojem, bo wiadomo było, że po fali entuzjazmu musi nadejść także epoka rozczarowań. Natomiast „neofici neurokomputingu" i zewnętrzni komentatorzy ich osiągnięć, to znaczy profesjonalni *naukoznawcy* oraz komentujący rozwój naukowy dziennikarze, nie znali umiaru i opisując kolejne osiągnięcia wyrażali podziw połączony ze zdumieniem. Ten podziw i zdziwienie trwają do dzisiaj, napisałem więc tę książkę po to, żeby Ci pomóc dzielić z innymi ten podziw, ale uwolnić Cię od zdumienia, bo pokażę Ci dokładnie, jak i dlaczego to wszystko działa.

Jeśli wahasz się jeszcze, czy czytać tę książkę, to powiem Ci coś jeszcze. Nie wiem, jaka jest dziedzina wiedzy, którą studiujesz lub chciałbyś studiować, ale zapewniam Cię, że bez trudu znajdziesz ciekawe zastosowanie sieci neuronowych właśnie w tej Twojej dziedzinie. Mogę Cię o tym solennie zapewnić, bowiem dla wielu takich obserwatorów zachowań sieci neuronowych wręcz zdumiewające jest to, jak bardzo ta technika okazała się **uniwersalna**. Możliwościami i zaletami sieci neuronowych zachwycają się równie entuzjastycznie inżynierowie, stosujący je na przykład do sterowania robotów, jak i bankowcy, którzy używali ich do ścigania defraudantów. Dla sieci neuronowych znajdowano zastosowania w astrofizyce przy modelowaniu początków Wszechświata i w przemyśle spożywczym przy wypiekaniu ciasteczek. W krótkim czasie okazało się, że sieci neuronowe da się zastosować bez mała do wszystkiego i w prawie każdej dziedzinie okazują się one skuteczniejsze niż tradycyjne metody komputerowe, używane w tych dziedzinach od wielu lat. Dla wielu tradycyjnych badaczy było to szokujące.

W historii rozwoju sieci neuronowych notowano wiele takich zdziwień. Jeśli sieć neuronowa potrafiła skuteczniej wspomagać oceny nowych metod uprawy roślin niż stosowane od lat obliczenia statystyczne – było to zaskoczeniem dla agrotechników. Jeśli sieć neuronowa potrafiła dokładniej kontrolować reakcję chemiczną niż model komputerowy oparty na bilansie energii i

masy – było to zdumiewające dla technologów. Jeśli prognoza podana przez sieć neuronową okazywała się trafniejsza od innych metod przy przewidywaniu przyszłych cen akcji lub kursów walut – było to niezrozumiałe dla ekspertów korzystających z metod tradycyjnej ekonometrii. Jeśli neurokomputer gwarantował lepszą kontrolę nad automatyzowanym procesem niż klasyczne procesory sygnałowe oraz najnowsze cyfrowe sterowniki, to fakt ten wzbudzał niepokój automatyków. Można byłoby bez końca wymieniać tych specjalistów, dla których sieci neuronowe otworzyły zupełnie nowe możliwości, ale też stały się źródłem zupełnie nowych wyzwań, bo ich dotychczasowa wiedza nagle okazywała się przestarzała. Nie pozwól, żeby postęp w dziedzinie sieci neuronowych Ciebie zaskoczył, tylko poznaj je jak najszybciej i jak najdokładniej – a ta książka bardzo Ci w tym pomoże!

Zadanie, jakie podjąłem pisząc tę książkę, nie jest łatwe. Wprawdzie już przystępując do pisania artykułów i pierwszej wersji książki miałem za sobą blisko 20 lat pracy badawczej przy tworzeniu i ciągłym doskonaleniu kolejnych sieci neuronowych – ale fachowe książki, których wcześniej napisałem na ten temat w sumie ponad 50[7], nie nadawały się do tego, żeby wyjaśnić fenomen sieci neuronowych w sposób prosty i przejrzysty dla niespecjalistów. Nie nadawały się do tego także moje liczne artykuły naukowe, jakie na ten temat publikowałem, gdyż wszystkie bez wyjątku były one adresowane do specjalistów-biocybernetyków i zakładały, że czytelnik ma sporą wiedzę na temat dyskutowanych w artykułach zagadnień. Tymczasem teraz miałem na ten sam temat napisać pracę **popularną**, czyli czytelną i zrozumiałą dla wszystkich. Wydawało się to prawie niewykonalne.

Jakże bowiem miałem oczekiwać od lękającego się matematyki lekarza, żeby poznawał silnie zmatematyzowaną i obiektywnie raczej trudną (nawet dla informatyków) teorię sieci neuronowych, zanim zacznie z nich korzystać podczas diagnozowania chorób lub przy planowaniu terapii dla swoich pacjentów? Albo czy mogłem namawiać dyrektora banku, który chciałby skorzystać z neuronowego doradcy przy udzielaniu kredytów, żeby wnikał w różne subtelności natury neuro-anatomicznej oraz neuro-fizjologicznej, stanowiące bazę pojęciową i koncepcyjną dla tworzenia sieci i dla ich wykorzystywania (a zwłaszcza uczenia)? Pisząc książkę podjąłem więc próbę wyjaśnienia, czym są sieci neuronowe bez użycia matematyki, i w całej niniejszej książce nie ma ani jednego wzoru matematycznego. Musiałem także wyjaśnić osobom, które o biologii słyszały niewiele i w dodatku bardzo dawno temu, w jakim zakresie i w jakim sensie sieci neuronowe można traktować jako modele rzeczywi-

[7] Pełna lista moich książek, artykułów i innych publikacji może być znaleziona w Internecie pod adresem: http://www.agh.edu.pl/uczelnia/tad/

stych struktur neuronowych (w szczególności fragmentów mózgu człowieka) i musiałem to zrobić bez przeprowadzania zbyt obszernego i zbyt wyczerpującego wykładu z zakresu anatomii i fizjologii systemu nerwowego. Nie mogłem przy tym zagłębiać się w szczegóły wiedzy na temat techniki sieci neuronowych, bo są one (jak wszelkie szczegóły każdej dziedziny wiedzy) trudne i skomplikowane, ale jednocześnie chciałem, żebyś po przestudiowaniu moich artykułów **naprawdę** wiedział, jak sieci neuronowe działają i dlaczego są tak bardzo użyteczne. Postanowiłem więc, że dam Ci możliwość samodzielnego **odkrywania** właściwości sieci neuronowych za pomocą coraz powszechniej dostępnych komputerów, z których z pewnością korzystasz – jeśli nie w domu, to przynajmniej w szkole albo w pracy. Napisałem więc serię **programów**, stanowiących integralną część tej książki, które pozwolą Ci samemu zbadać i **odkryć**, czym są i jak działają sieci neuronowe.

Trud pisania tych programów był spory, ale konieczny. Być może zdziwiłem Cię tym stwierdzeniem, bo wiesz dobrze o tym, jak wiele fajnych programów na różne tematy jest w Internecie, więc może zamiast pisać własne programy – lepiej było dobrze poszukać w Sieci?

Oczywiście znam i cenię zasoby, jakie na temat sieci neuronowych zgromadzono i udostępniono w Internecie. Dlatego jeszcze w jednym z artykułów cyklu tworzonego w ***Enterze*** podałem długi wykaz znanych mi programów komputerowych, umożliwiających budowę i symulację na komputerach klasy PC prostych sieci neuronowych – oraz źródeł, z których te programy można było pozyskać. Niestety, nie było to dobre rozwiązanie. Programy te wymagały jednak zwykle od użytkowników posiadania większej wiedzy, niż ja byłem w stanie zapewnić moim Czytelnikom poprzez napisanie popularnych artykułów, a ponadto z reguły udostępniane były według takiej zasady, że dla uzyskania ich pełnej funkcjonalności konieczne było wykupienie licencji (na co młodzieży szkolnej i innych hobbystów z reguły nie było stać), zaś za darmo można było korzystać wyłącznie z bardzo ograniczonych wersji tych programów (typu *demo*). W sumie stosowanie tych programów, które były dostępne, dalekie było od tego, co ja chciałem zapewnić moim Czytelnikom – to znaczy radości samodzielnego eksperymentowania z samodzielnie budowanymi sieciami neuronowymi.

W tej sytuacji trzeba było napisać samemu programy, którymi Czytelnik mógłby się pobawić, zdobywając obok wiedzy teoretycznej także i praktyczne doświadczenia związane z problematyką sieci neuronowych. Uczyniłem to i dołączyłem te programy do poprzedniej książki (zatytułowanej *Elementarne wprowadzenie do sieci neuronowych z przykładowymi programami*) w formie dyskietki. Obecnie programy te są powszechnie dostępne na stronie ***http://www.agh.edu.pl/dydaktyka/sieci_neuronowe/basic/*** i wiele osób z nich korzysta.

Niestety, tamta wersja tych programów została opracowana w języku Basic. Ściślej mówiąc – w jego dialekcie używanym przez interpreter[8] QBASIC. Interpreter ten odszedł jednak w zapomnienie wraz ze zmierzchem systemu operacyjnego MS-DOS, a więc już dość dawno. Patrząc z perspektywy czasu, można byłoby uznać, że było to podówczas dobre rozwiązanie. Przede wszystkim, programy napisane w „starym, dobrym BASIC-u" mają to do siebie, że – jak w przypadku każdego języka *interpretowanego* – program jest po prostu plikiem tekstowym, który możesz obejrzeć (i zmodyfikować!) w dowolnym edytorze tekstu. Niestety, nie ma róży bez kolców. Do uruchomienia takiego programu wymagany jest wspomniany interpreter – program QBASIC. Tak, tak – jeśli do tej pory nie byłeś świadomy faktu, że komputer tak naprawdę nie może po prostu wykonać pliku tekstowego, choćby zawierał on tekst programu napisanego w dowolnie wymyślonym języku programowania – to właśnie zostałeś wyprowadzony z błędu. Systemy typu Windows czy Linux nie mają obecnie wbudowanego interpretera QBASIC (co kiedyś było normą), więc musisz go sobie zainstalować. Jest on także dostępny wraz ze swoim podręcznikiem (jako pliki o nazwie QBASIC.EXE oraz QBASIC.HLP) na wskazanej wyżej stronie: ***http://www.agh.edu.pl/dydaktyka/sieci_neuronowe/basic/,*** jednak QBASIC był niestety programem prymitywnym i jego możliwości są mocno ograniczone. Dziś, kiedy normą dla użytkownika są programy oferujące bogaty, przyjazny użytkownikowi interfejs okienkowy, używanie zarówno samego środowiska QBASIC, jak i programów w nim napisanych, może być irytujące.

Programy przeznaczone to tego, żeby ułatwić i umilić czytanie **tej książki**, są napisane w zupełnie innej technologii. Zostały one napisane w języku C#, który jest językiem *kompilowanym*. Oznacza to, że dostaniesz do ręki gotowe programy, które możesz zainstalować na swoim komputerze i uruchomić kilkoma kliknięciami – tak jak większość innych programów. Ponieważ zostały napisane (przez współautorów tej książki, **Tomasza Gąciarza, Barbarę**

[8] Interpreter jest programem służącym... do wykonywania innych programów. Jeśli wydaje Ci się to trochę skomplikowane – nie przejmuj się, nie musisz się na tym znać, aby czytać tę książkę i aby korzystać z dołączonych do niej programów. Jeśli jednak Cię to interesuje, wiedz, że programy dzielą się na dwie kategorie:

- **interpretowane**, w których program jest zapisany w pliku tekstowym, a jego uruchomienie polega na odczytaniu go przez interpreter i wykonaniu krok po kroku;
- **kompilowane**, w których tekst programu (tzw. kod źródłowy) ma również postać pliku tekstowego, ale jest on „raz a dobrze" przekształcany na format bezpośrednio zrozumiały dla procesora komputera (tzw. kod wynikowy), zapisywany najczęściej w pliku z rozszerzeniem „.exe". Brzmi znajomo, prawda? Tak jest, większość programów na dysku Twojego komputera to najprawdopodobniej właśnie programy kompilowane.

**Borowik** oraz **Bartosza Lepera**) specjalnie z myślą o systemie Windows, mają one łatwy w obsłudze i przejrzysty interfejs. Pogramy te możesz sobie ściągnąć (legalnie i za darmo!) z mojej strony internetowej ***http://www.agh.edu.pl/tad***.

Dokładniejsze informacje na temat, jak te programy ściągnąć i jak ich używać, podane będą w rozdziale 4.

Pojawia się jednak uzasadnione pytanie: Dlaczego, mimo że tekst programów w BASIC-u można było obejrzeć „gołym okiem" i samemu zrozumieć, co i jak on robi, zaś skompilowane programy napisane pierwotnie w C# nie nadają się zbytnio do „czytania" – zdecydowaliśmy się na użycie tej nowej technologii?

Przede wszystkim dlatego, że nowe programy są **prostsze w użyciu**. Co więcej, ich używanie dostosowane jest do nawyków i upodobań nowoczesnych rozgrymaszonych użytkowników komputerów (to Ty jesteś takim nowoczesnym rozgrymaszonym użytkownikiem komputera, nie wypieraj się!), przyzwyczajonych do operowania programami za pomocą udogodnień dostarczanych im przez system Windows: wygodne i szybkie wybory z rozwijalnych menu, różne pomysłowe kontrolki, przyciski do klikania myszką itp. – a nie pracowite wpisywanie danych z klawiatury, co było jedyną formą komunikacji z programami w BASIC-u!

To zaś, że te nowoczesne i udoskonalone programy są skompilowane (a więc nieczytelne), nie powinno stanowić większego problemu: w końcu i tak większość Czytelników będzie po prostu chciała te przykładowe programy uruchomić i poeksperymentować z nimi, a nie badać, jak są zbudowane. Tych nielicznych pasjonatów, którzy chcą zajrzeć im „pod maskę", będzie siłą rzeczy mniej. **Jednak nawet i oni nie zostaną w tej książce pominięci!** Na podanej wyżej stronie internetowej ***http://www.agh.edu.pl/tad*** znajduje się także pełny kod źródłowy wszystkich napisanych przez nas programów, a także zestaw narzędzi pozwalających te kody przeglądać i analizować. Dla odważniejszych i bardziej doświadczonych są tam także narzędzia pozwalające kody naszych programów modyfikować, kompilować i samodzielnie wykonywać. Wykorzystane do tego jest darmowe środowisko programistyczne firmy Microsoft – Visual Studio Express 2005. Również tych, którzy chcą tylko obejrzeć kod źródłowy naszych programów, zachęcamy do instalacji tego **darmowego** środowiska na swoich komputerach – dzięki niemu korzystanie z programów ilustrujących wybrane zagadnienia opisywane w książce będzie o wiele łatwiejsze!

Jak więc widzisz, książkę i zasoby umieszczone we wskazanym miejscu w Internecie możesz wykorzystywać na trzy sposoby:

- Jeśli interesują Cię tylko sieci neuronowe od strony teoretycznej i nie masz ochoty bawić się jakimikolwiek programami – wystarczy, że przeczytasz treść książki.
- Jeśli lubisz sam sprawdzić, jak to i owo działa, to możesz ściągnąć sobie gotowe programy, które pozwolą Ci budować i badać **na Twoim komputerze** sieci neuronowe. W ten sposób połączysz teorię (przeczytaną w książce) z praktyką (wynikającą z zabawy z naszymi programami), nabierając dzięki temu **dwojakich** umiejętności związanych z teorią sieci neuronowych i z metodami ich stosowania.
- Jeśli jednak jesteś dodatkowo amatorem i pasjonatem programowania – to będziesz mógł obejrzeć dokładnie, jak te nasze programy zostały zbudowane i będziesz mógł je do woli poprawiać, zmieniać i doskonalić. My nie mamy tu nic do ukrycia!

Dzięki temu, że zadasz sobie trud przeczytania (a może także przerobienia) tekstów naszych programów, będziesz mógł zrozumieć, jak one działają, a także będziesz mógł zmienić coś tu i ówdzie. Jeśli zechcesz zmodyfikować kod źródłowy naszych programów i spróbujesz sprawdzić, „co by było, gdyby…", to wejdziesz do elitarnego grona **twórców** sieci neuronowych, znacznie bardziej zaszczytnego od grona samych tylko **użytkowników** tychże sieci.

Zapraszam więc wszystkich chętnych do tego, by zechcieli zagłębić się w kolejnych rozdziałach tej książki i za jej pośrednictwem (a także przy wykorzystaniu skojarzonych z książką programów) poznali fascynujący świat sieci neuronowych – głównie jako narzędzi sztucznej inteligencji, które miały i mają bardzo wiele zastosowań informatycznych, ale także jako ciekawych modeli fragmentów naszego własnego mózgu, które być może pomogą zrozumieć tajemniczy świat naszej własnej inteligencji oraz naszej skomplikowanej psychiki.

# 1. Wprowadzenie do wiedzy o sieciach neuronowych naturalnych i sztucznych oraz o ich zastosowaniach

## 1.1. Dlaczego warto poznać sieci neuronowe?

Ta książka przeznaczona jest na to, żebyś się dowiedział, drogi Czytelniku, w sposób łatwy, miły i przyjemny, czym są sieci neuronowe, w jaki sposób i dlaczego działają, a także do czego i jak można je stosować. Skoro wziąłeś tę książkę do rąk, to zapewne tego właśnie pragniesz się dowiedzieć. Książka jest jednak gruba, a więc jej przestudiowanie będzie pewnie wymagało sporego wysiłku. Możesz więc postawić sobie pytanie: Czy warto to robić, a jeśli tak, to dlaczego? Może lepiej odłożyć tę książkę i zająć się jakąś grą komputerową?

Najprostsza odpowiedź na to pytanie brzmi: ***Tak, warto*** *poznać sieci neuronowe, bo interesuje się nimi wielu badaczy i praktyków, bo z ich pomocą dokonano już wielu ciekawych odkryć i zapewne będą one źródłem wielu dalszych osiągnięć.*

Jeśli te powody jeszcze Cię nie przekonały, że warto poświęcić trochę czasu na zapoznanie się z sieciami neuronowymi, to może uznasz, że warto je poznać z tego powodu, że ciągle się o nich słyszy, bo są po prostu **modne?**

Czy jednak są to wystarczające powody? Wszak w informatyce od lat obserwujemy różne mody i związane z nimi przypływy i odpływy zainteresowania różnymi szczegółowymi zagadnieniami, wpływające na kierunki prac badawczych, kształtujące obraz rynku komputerowego i ogniskujące na określonych zagadnieniach wysiłek licznych programistów. Czas trwania takiej mody bywa różny, przeciętnie szacować go można na okres od kilku miesięcy do kilku lat. Na ogół moda taka trwa do czasu, aż kolejna fascynacja nie ściągnie rzeszy „poszukiwaczy złota" w zupełnie nowe rejony. Przykładów takich modnych tematów można wymienić bez liku: niedawno przeżywaliśmy masową fascynację ***Internetem*** (zresztą „atak" na inteligentne ***przeglądarki*** in-

ternetowe trwa nadal), mówi się obecnie wiele o technikach obliczeń ***gridowych***, wciąż widoczna jest fala mody na automaty ***komórkowe*** oraz na tzw. techniki ***agentowe***, mają swoich wiernych „fanów" ***fraktale*** i ***chaos***, spotyka się nawracające okresowo (jak epidemie dżumy!) powszechne fascynacje algorytmami ***genetycznymi*** albo teorią zbiorów ***rozmytych***...

Można powiedzieć, że w informatyce zmienne mody są niezmiennie modne!

Od początku lat 90. zapanowała moda na sieci neuronowe – i trwa do dzisiaj. Nieco więcej o tej modzie napisałem w przedmowie, dlatego jeśli jeszcze jej nie przeczytałeś (a znam takich, którzy z zasady pomijają wszelkie przedmowy…), to radzę, żebyś cofnął się o kilka kartek i jednak **tę konkretną** przedmowę przeczytał. Jest w niej bowiem wiele ciekawych, a także ważnych i użytecznych wiadomości, których w dalszej części książki nie będę ponownie powtarzał, a które na pewno Ci się przydadzą, jeśli na poważnie chcesz się zająć sieciami neuronowymi jako obiektem Twojego zainteresowania – a może także jako narzędziem Twojej przyszłej pracy.

Teraz chcę Ci powiedzieć coś ważnego o tym rozdziale.

**Otóż jeśli nie interesują Cię biologiczne podstawy budowy i funkcjonowania sieci neuronowych – to możesz go całkowicie pominąć.**

Naprawdę!

W następnych rozdziałach pokażę Ci, jak możesz budować sztuczne sieci neuronowe i jak możesz ich używać. Tamte rozdziały **musisz** czytać po kolei, bo jak coś pominiesz, to będziesz miał trudności ze zrozumieniem dalszego ciągu przedstawianych w książce rozważań.

Ale ten rozdział jest inny. Mówi on o tym, jak ludzie odkryli sieci neuronowe badając swój własny mózg. Badania te uczeni prowadzili od lat, po to żeby odkryć tajemnice ludzkiej inteligencji, a nagle się okazało, że ich wyniki mogą być użyteczne w informatyce. Ten rozdział opowie Ci także o tym, jak te „wypożyczone od biologii" sztuczne sieci neuronowe pomagają dzisiaj odkrywać kolejne, dalsze tajniki tego mózgu. Ja to uważam za niesłychanie ciekawe, dlatego napisałem o tym tak dużo – ale jeśli Ty chciałbyś jak najszybciej zacząć sam odkrywać tajemnice sieci neuronowych, to naprawdę możesz cały ten rozdział pominąć. Mam nadzieję, że gdy zobaczysz, jak te sieci neuronowe fajnie działają – to wrócisz do tego rozdziału, żeby się więcej dowiedzieć o ich genezie, ale pamiętaj: **nie musisz tego robić.**

Jeśli czytasz dalej, to znaczy, że jesteś ciekaw tego, jak doszło do odkrycia sieci neuronowych, a ja spróbuję tę Twoją ciekawość zaspokoić. Wiesz już, że sieci neuronowe stanowią bardzo uproszczony (przez co łatwiejszy do ogarnięcia myślą lub do zamodelowania na komputerze), ale zaskakująco bogaty i ciekawy model rzeczywistego biologicznego systemu nerwowego. Można

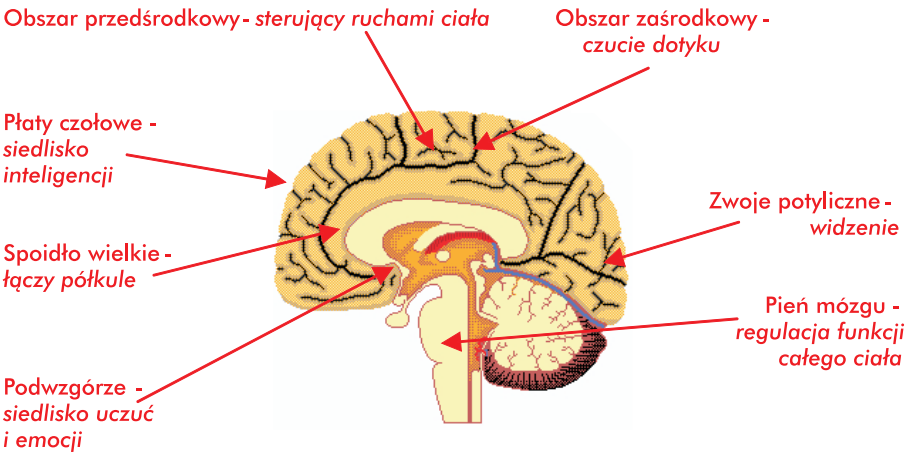
Rys. 1.1. Ludzki mózg – pierwowzór i niedościgły ideał dla badaczy sieci neuronowych

w skrócie powiedzieć, że sieci neuronowe są po prostu uproszczonymi modelami pewnych fragmentów naszego własnego mózgu (rys. 1.1).

Jak doszło do ich zbudowania?

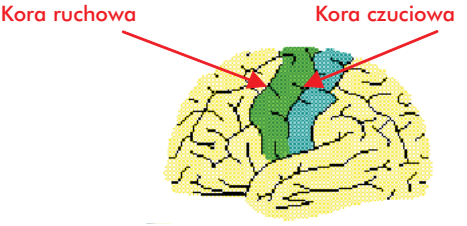
## 1.2. Co było wiadome o mózgu, gdy zaczęto budować pierwsze sieci neuronowe?

Wnętrze mózgu zawsze fascynowało ludzi, ale mimo wielu lat intensywnych badań nie zdołano do tej pory do końca wyjaśnić ani do końca zrozumieć wszystkich fenomenów związanych z jego działaniem. Dopiero ostatnie lata przyniosły w tym zakresie zasadniczy postęp (będziemy jeszcze o tym mówić dodatkowo w następnym podrozdziale), natomiast w latach 90. XX wieku, kiedy najszybciej i najintensywniej rozwijały się podstawy sieci neuronowych, wiadomości na ten temat było relatywnie mało. Wiedziano już wtedy, gdzie zlokalizowane są w mózgu najważniejsze ośrodki związane z podstawowymi funkcjami motorycznymi, percepcyjnymi i intelektualnymi (rys. 1.2).



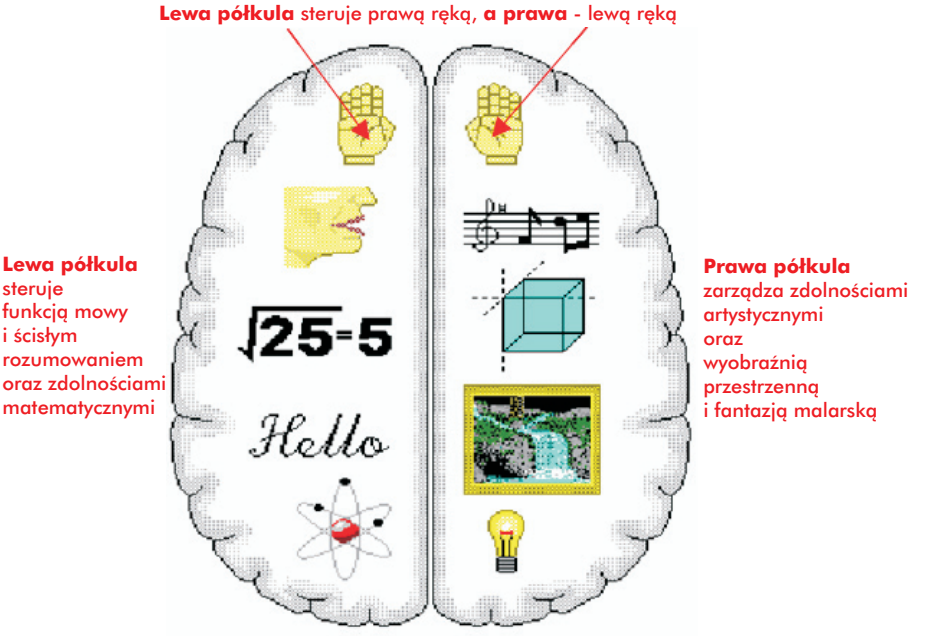

Rys. 1.2. Widok mózgu człowieka w przekroju podłużnym z zaznaczeniem lokalizacji najważniejszych funkcji

Jednak wiedza na temat funkcjonowania poszczególnych elementów mózgu była dość ogólnikowa. Wiedziano dość dobrze, jak funkcjonują te fragmenty mózgu, które odpowiedzialne są za sterowanie ruchem lub za odbiór podstawowych wrażeń zmysłowych (rys. 1.3), ponieważ liczne choroby, wypadki lub rany wojenne pozwoliły stwierdzić, jakie ubytki ruchu (paraliż) albo czucia towarzyszą uszkodzeniom określonych konkretnych fragmentów mózgu.




Rys 1.3. Lokalizacja funkcji sterowania ruchem i prostego czucia w mózgu

Jednak pytanie o naturę i lokalizację wyższych czynności psychicznych prowadziło nieuchronnie na owym dość prymitywnym jeszcze etapie wiedzy do dość ogólnikowych informacji, głównie sprowadzających się do tego, że istnieje dość wyraźna specjalizacja poszczególnych półkul mózgowych (rys. 1.4). Wynikało to w dużej mierze z faktu, że względy etyczne nie pozwalały




Rys. 1.4. Podział zadań pomiędzy półkule mózgowe (w znacznym stopniu hipotetyczny)

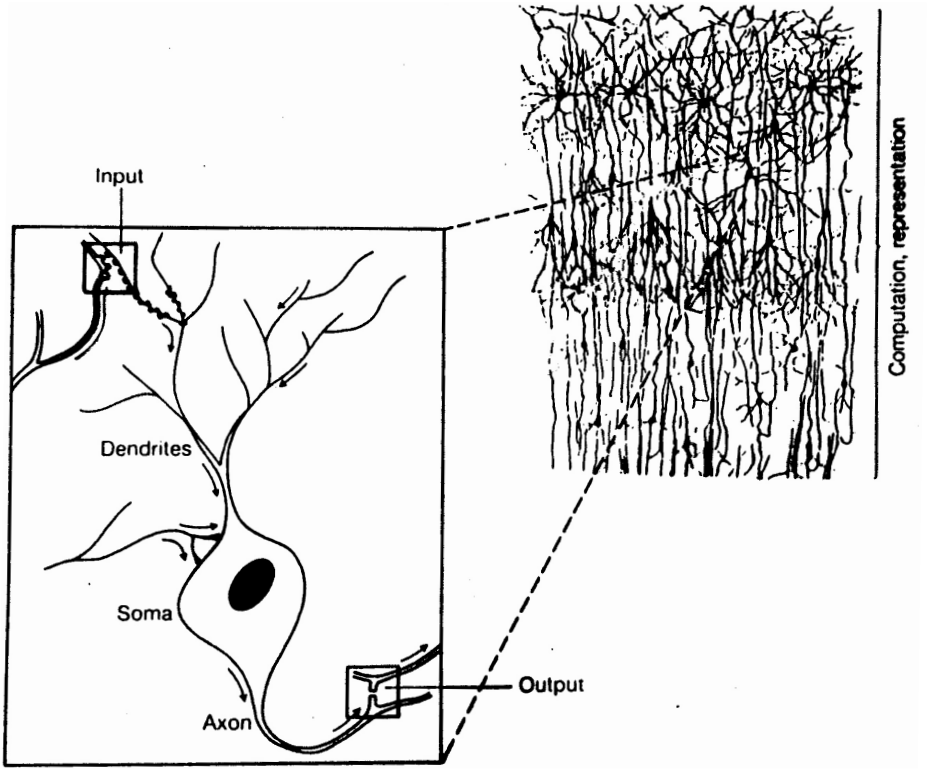
na prowadzenie eksperymentalnych badań na ludzkim mózgu. Czym innym była bowiem wspomniana wyżej praktyka skrzętnego zbierania i analizowania wszelkich informacji o tym, jakie zmiany czynnościowe, psychologiczne i morfologiczne obserwowane w odczuciach i zachowaniu ludzi towarzyszą określonym uszkodzeniom mózgu powstającym w sposób naturalny (nie związany z tymi badaniami), a czym innym byłoby celowe manipulowanie elektrodą lub skalpelem w tkance zdrowego mózgu w celu zdobycia wiadomości, jakie są funkcje poszczególnych jego części.

Oczywiście, można było w tamtych latach przeprowadzać doświadczenia na zwierzętach – i robiono to bardzo często. Jednak mordowanie niewinnych zwierząt w imię zaspokajania naszej ciekawości, nawet jeśli jest to szlachetna ciekawość naukowca, miało zawsze dość dwuznaczny charakter moralny. W dodatku z doświadczeń na zwierzętach nie dało się bezpośrednio wyprowadzać wniosków o zachowaniu i własności ludzkiego mózgu, gdyż w tym zakresie dystans, jaki nas dzieli od zwierząt, jest jednak większy niż w przypadku badań mięśni, serca czy krwi.

Czym więc dysponowali pierwsi twórcy sieci neuronowych, którzy chcieli swoje konstrukcje wyposażyć w możliwie dużo cech i właściwości wzorowanych na funkcjonowaniu prawdziwego biologicznego mózgu?

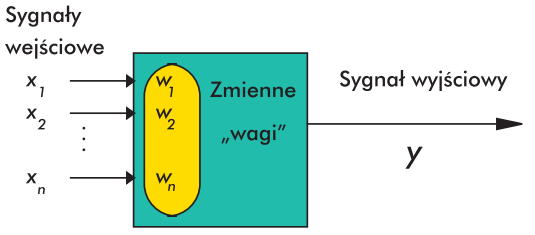
Otóż po pierwsze wiedzieli, że mózg składa się z oddzielnych komórek (neuronów) pełniących rolę biologicznych procesorów. Jako pierwszy opisał mózg człowieka jako sieć powiązanych ze sobą, ale w miarę autonomicznych elementów anatom hiszpański Ramón y Cajal (nagroda Nobla w 1906 roku – patrz tabela umieszczona w przedmowie). On też wprowadził koncepcję **neuronów**, wyspecjalizowanych komórek przetwarzających informacje, odbierających i analizujących wrażenia zmysłowe, a także wypracowujących i wysyłających sygnały sterujące do tych wszystkich elementów, które w ciele człowieka znajdują się pod kontrolą mózgu (mięśni sterujących ruchami ciała, gruczołów oraz innych narządów wewnętrznych). O budowie neuronu powiemy więcej w rozdziale 2., ponieważ jego techniczny odpowiednik jest głównym składnikiem rozważanych tam struktur sieci neuronowych, natomiast na rysunku 1.5 można zobaczyć, w jaki sposób z gęsto utkanej sieci neuronów tworzących korę mózgową udało się wyodrębnić pojedynczą komórkę, stanowiącą, jak wspomniałem, skomplikowany biologiczny procesor odpowiedzialny za wszystkie funkcje i za wszystkie działania naszego systemu nerwowego (nie tylko zresztą mózgu, ale również na przykład elementów sympatycznego i parasympatycznego systemu sterującego pracą wszystkich narządów wewnętrznych).

O neuronach wiedziano w momencie budowania pierwszych sieci neuronowych dosyć dużo, gdyż u niektórych gatunków zwierząt (na przykład




Rys. 1.5. Kluczem do stworzenia koncepcji sieci neuronowych było wyodrębnienie z tkanki nerwowej poszczególnych komórek (neuronów), będących biologicznymi procesorami

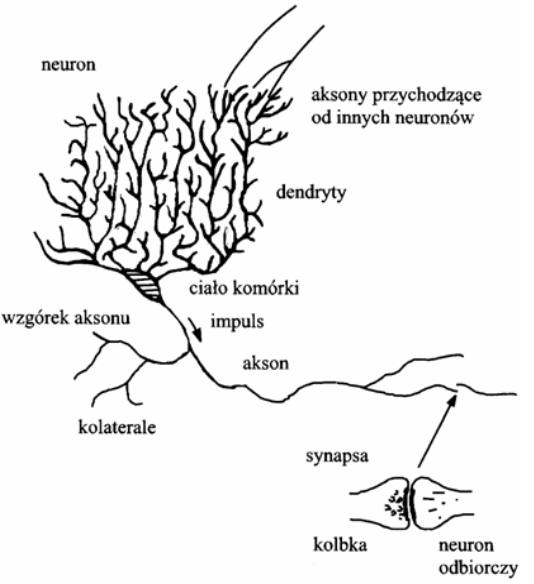
u kalmara *Loligio*) są one tak duże, że zręcznym badaczom (Hodgkin i Huxley, nagroda Nobla w 1963 roku) udało się zbadać zmiany biochemiczne i bioelektryczne, jakie w nich zachodzą podczas przesyłania i przetwarzania sygnałów będących nośnikami informacji nerwowej. Jednak kluczowe okazało się stwierdzenie, że opis rzeczywistego neuronu (w istocie bardzo skomplikowanego) można bardzo mocno uprościć, redukując zasady przetwarzania informacji, jakie w nim znajdują zastosowanie do kilku prostych zależności (opisanych w rozdziale 2.). Taki skrajnie uproszczony neuron, pokazany sche-




Rys. 1.6. Struktura i podstawowe elementy sztucznego neuronu

matycznie na rysunku 1.6, nadal pozwala tworzyć sieci o ciekawych i użytecznych właściwościach, a jednocześnie można go bardzo tanio zbudować.

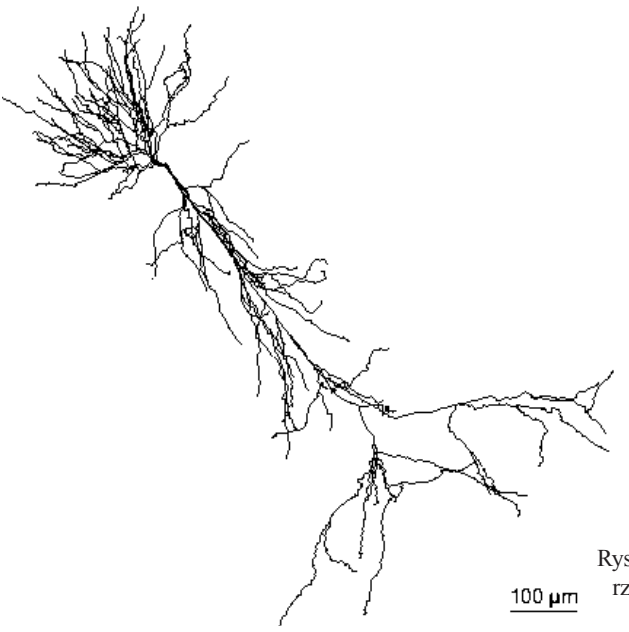
Elementy odnotowane na rysunku 1.6 (zwłaszcza tajemnicze „wagi", zaznaczone wewnątrz bloku symbolizującego neuron) będą dokładnie omawiane w kolejnych rozdziałach książki, więc chwilowo nie zaprzątaj sobie nimi głowy. Natomiast zauważ jedno: prawdziwy biologiczny neuron ma niesłychanie bogatą i urozmaiconą budowę (por. rysunek 1.7). Jego techniczny odpowiednik pokazany na rysunku 1.6 jest bardzo mocno uproszczony co do struktury, a jeszcze mocniej jest uproszczony w związku z czynnościami, do jakich jest zdolny. Jednak nie przeszkadza to nam uzyskiwać w sieciach sztucznych neuronów tak złożonych i ciekawych zachowań, jak te, które będę Ci opisywał w dalszych rozdziałach tej książki. Jakże bogate i różnorodne możliwości ma więc oryginalna biologiczna sieć, neuronowa, budująca nasz mózg!




Rys. 1.7. Rysunek pokazujący (w uproszczeniu!) kształt rzeczywistej komórki nerwowej

Neuron pokazany na rysunku 1.7 jest wynikiem fantazji grafika, natomiast na rysunku 1.8. możesz zobaczyć przykładowy wygląd rzeczywistej komórki nerwowej, wypreparowanej z mózgu szczura – ale neurony człowieka wyglądają prawie identycznie. Prawda, że taki biologiczny neuron ma naprawdę skomplikowaną budowę?

Sztuczne neurony właśnie dlatego, że są tak bardzo uproszczone, mogą być w miarę łatwo i tanio zrealizowane technicznie w postaci prostego układu elektronicznego (pierwsze sieci neuronowe budowano jako specjalne elektroniczne maszyny, zwane *perceptronami*) albo można je również łatwo za-



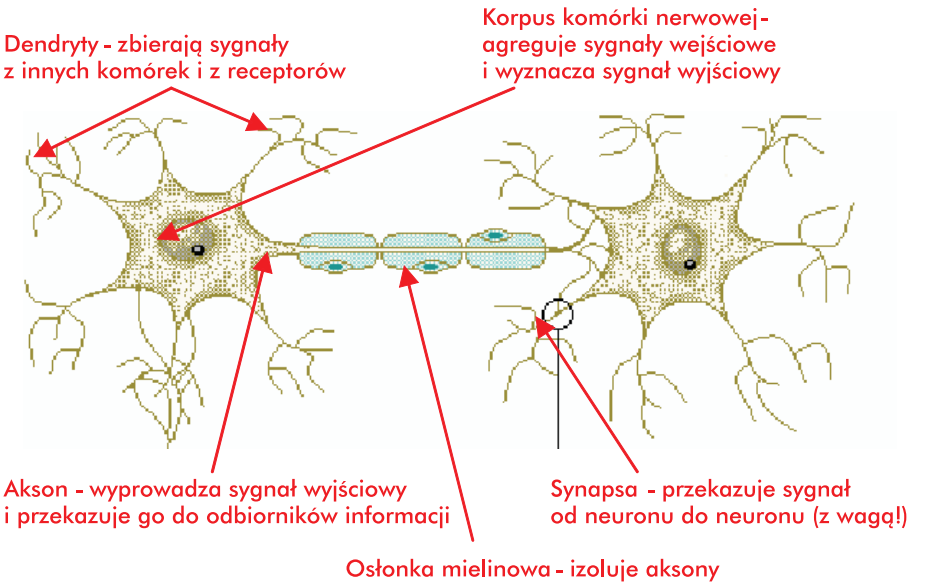

Rys. 1.8. Preparat mikroskopowy rzeczywistego neuronu z kory mózgowej szczura

modelować w formie programu symulującego działanie takiej komórki na typowym komputerze klasy PC (lub dowolnym innym). W systemach obecnie wykorzystywanych praktycznie z reguły wybiera się właśnie realizację symulacyjną, która daje wygodne i tanie narzędzie, pozwalające symulować zarówno pojedyncze neurony, jak i całe ich sieci.

## 1.3. Jak budowano pierwsze sieci neuronowe?

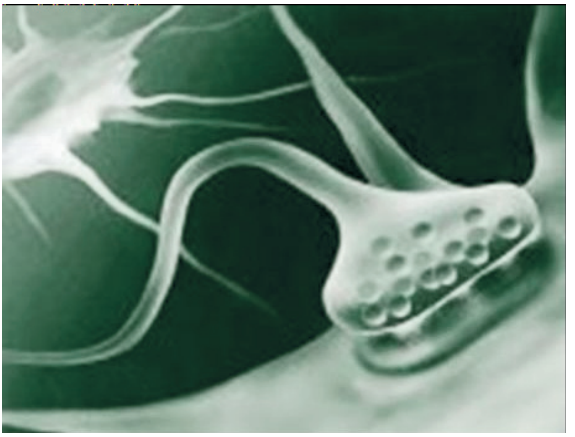
Wróćmy jednak od wyliczania informacji biologicznych, na których opierali się twórcy pierwszych sieci neuronowych, do pokazania tego, co neurocybernetycy z tymi informacjami biologicznymi zrobili, żeby uzyskane sieci mogły być w miarę tanio budowane i wygodnie wykorzystywane. Mieli oni już całkiem sporą wiedzę na temat tego, **jakie czynności** potrafi wykonywać biologiczny neuron. Wystarczy zajrzeć do tabeli P.1 w przedmowie, żeby stwierdzić, jak wiele nagród Nobla przyznano w ciągu całego XX stulecia za odkrycia, które pośrednio lub bezpośrednio dotyczyły właśnie komórki nerwowej i jej funkcjonowania. Najważniejsza wiadomość, jaką udało się uzyskać biologom, dotyczyła miejsc, w których jeden neuron przekazuje sygnał do innego neuronu (rys. 1.9). To było zadziwiające i fascynujące. Okazało się, że przy przetwarzaniu informacji w mózgu bynajmniej nie są najważniejsze wielkie i skomplikowane ciała komórek ani długie i rozkrzewione włókna nerwowe

(aksony i dendryty), komunikujące te neurony ze sobą. Najważniejsze okazały się systemy pośredniczące w przekazie informacyjnym pomiędzy neuronami, nazywane *synapsami.* Są one bardzo małe, tak małe, że rozdzielczość typowo używanych w biologii mikroskopów optycznych była zbyt mała, żeby te struktury ujawnić i opisać. Na rysunku 1.9 też ledwie je widać.




Rys. 1.9. Najmniejszy funkcjonalny fragment systemu nerwowego tworzą dwa połączone ze sobą i współpracujące neurony. W tej strukturze najważniejsza jest synapsa łącząca te dwa neurony

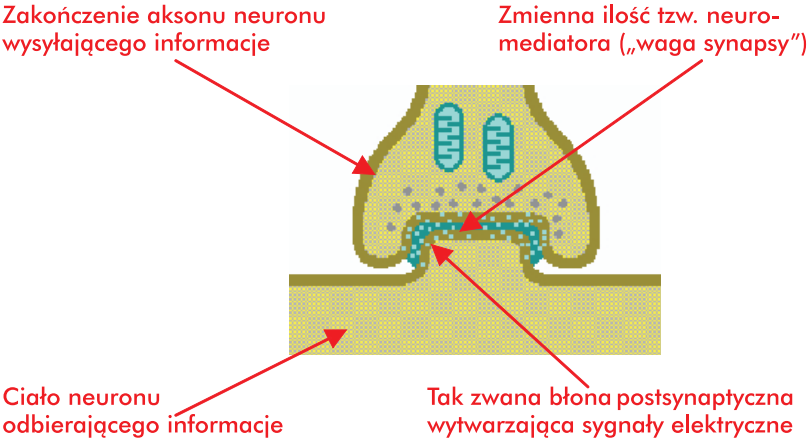
Dopiero mikroskopy elektronowe pokazały, jak złożone i ciekawe kształty mają w rzeczywistości synapsy (rys. 1.10), zaś seria genialnych eksperymentów neurofizjologa brytyjskiego, Johna Ecclesa, udowodniła, że przy przechodzeniu sygnału nerwowego przez synapsę zaangażowane są specjalne substancje chemiczne, tak zwane neuromediatory, które uwalniane w zakończeniu aksonu neuronu przekazującego infomację docierają do tak zwanej błony postsynaptycznej należącej do neuronu, który informacje odbiera (patrz bardzo uproszczony schemat budowy synapsy, pokazany na rysunku 1.11). W największym uproszczeniu można powiedzieć, że uczenie neuronu (i całego mózgu) polega na tym, że taki sam sygnał przysłany poprzez akson od komórki nadającej wiadomości może uwalniać do synapsy większą albo mniejszą ilość neuromediatora. Jeśli w wyniku uczenia okazuje się, że sygnał jest ważny – to ilość neuromediatora się zwiększa. Jeśli ustalono, że sygnał jest nieważny, to ilość neuromediatora maleje. To właśnie na tym polega tajemni-



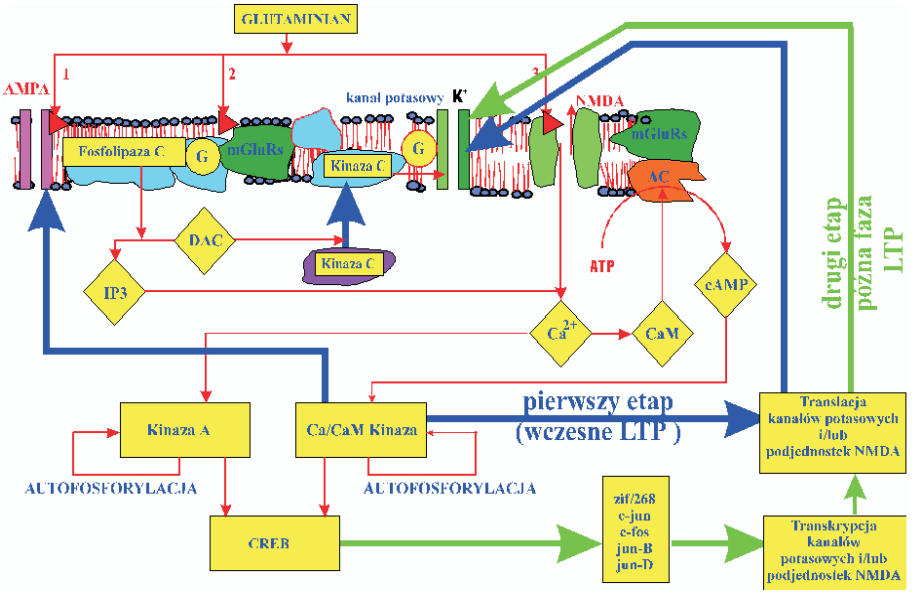
Rys. 1.10. Najważniejszy element neuronu, mający zasadniczy wpływ na procesy uczenia się sieci neuronowych: synapsa nerwowa

ca uczenia się i pamięci, chociaż jeszcze raz chcę podkreślić, że naszkicowany przed chwilą mechanim jest krańcowo uproszczony w stosunku do złożonych biologicznych procesów, które naprawdę mają miejsce w synapsie. Najlepszym dowodem na to, z jak ważnym i trudnym procesem mamy tu do czynienia, jest fakt, że za odkrycie tego, jak synapsy przenoszą informacje od neuronu do neuronu, a także za wykrycie mechanizmu zmian zachodzących w synapsach, gdy mózg się uczy i zdobywa nowe informacje, John Eccles otrzymał w 1963 roku nagrodę Nobla (patrz tabela zamieszczona w Przedmowie).

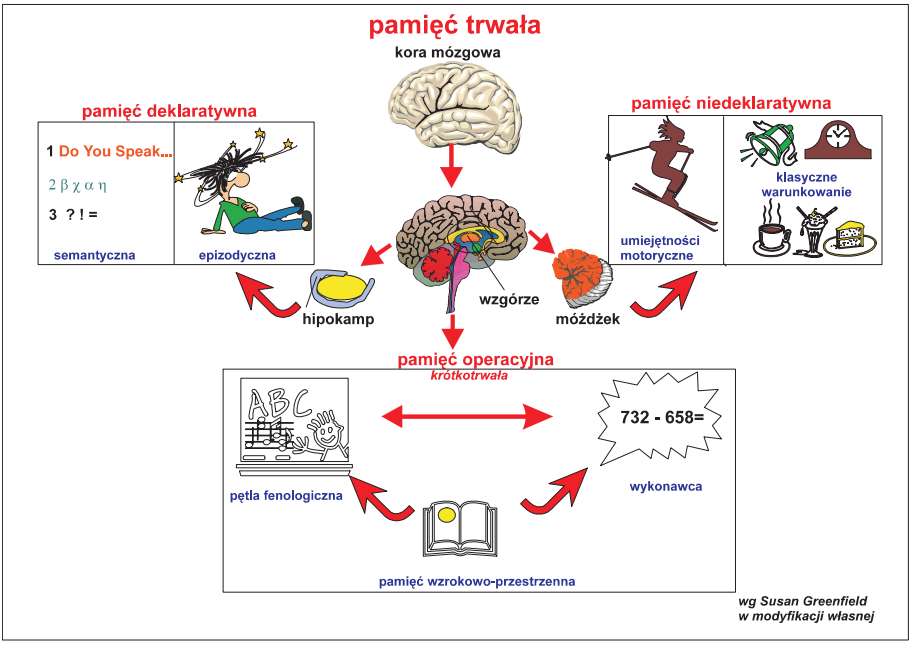
Specjaliści od sieci neuronowych skwapliwie wykorzystali te informacje i dzięki temu budowane przez nich systemy zyskały jako jedną z najważniej-



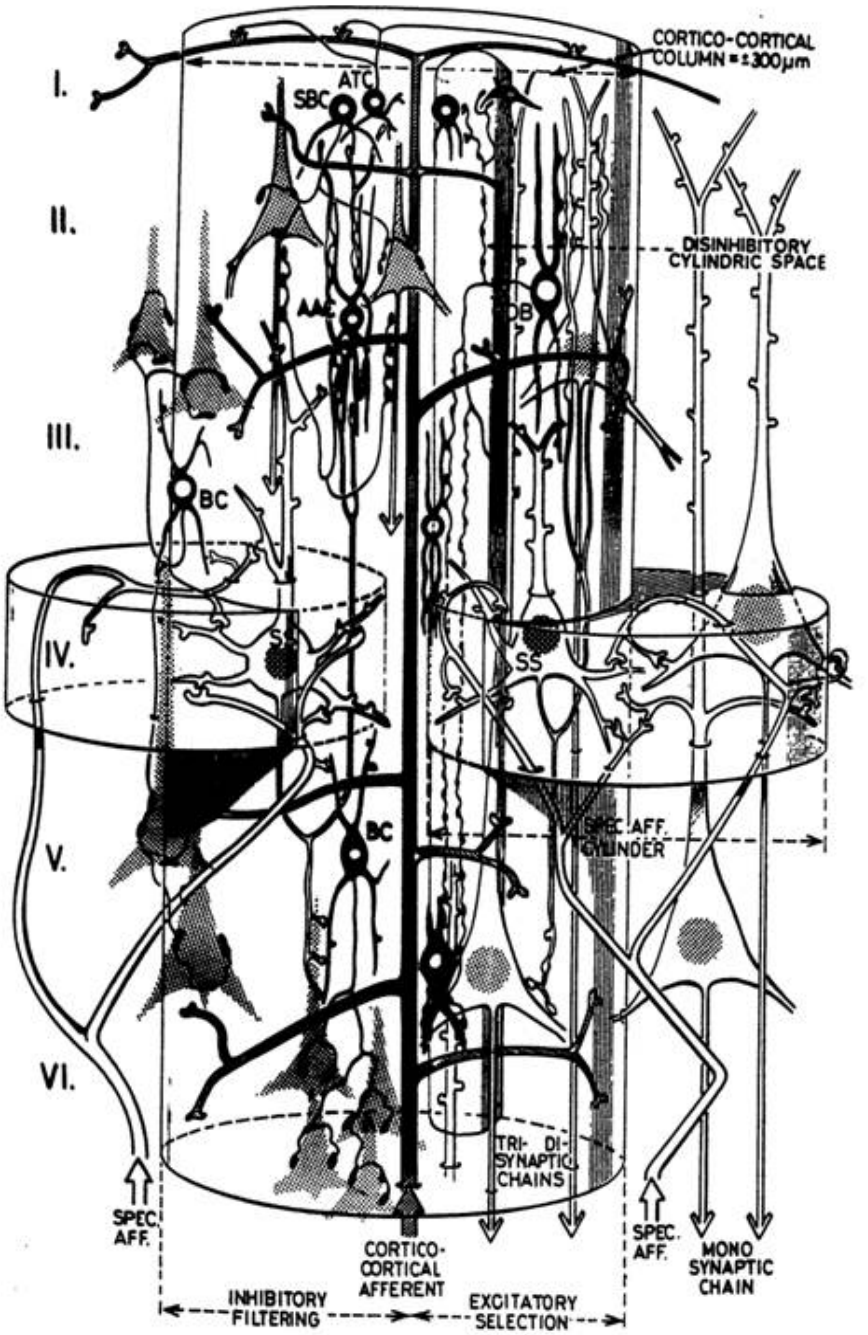

Rys. 1.11. Bardzo uproszczony schemat budowy synapsy pozwala zobaczyć, gdzie się mieści jej „waga", o której w sieciach neuronowych mówi się właściwie bezustannie




Rys. 1.12. Mechanizmy biochemiczne zaangażowane w proces zapamiętywania wiedzy są raczej skomplikowane, ale w sieciach neuronowych są odtwarzane w sposób uproszczony




Rys. 1.13. Z różnych rodzajów pamięci, jakie psychologowie wyróżniają u człowieka, w sieciach neuronowych odwzorowuje się wyłącznie jedną: pamięć niedeklaratywną




Rys. 1.14. Przykładowy rysunek przestrzennej rekonstrukcji połączeń elementów systemu nerwowego

szych swoich właściwości: **zdolność uczenia się.** Oczywiście, znowu biologiczna zasada uczenia, która obiektywnie jest raczej dosyć skomplikowana i odwołuje się do bardzo złożonych procesów biochemicznych (rys. 1.12), musiała zostać bardzo uproszczona, żeby można było uzyskać narzędzie, które dałoby się skutecznie wykorzystać jako system do rozwiązywania praktycznych problemów informatycznych. Co więcej, trzeba było się zdecydować, że przedmiotem uczenia w sieciach neuronowych będzie jedynie ten rodzaj wiedzy, który psychologia zalicza do funkcji **pamięci niedeklaratywnej**, która jednak, jak wiadomo, nie jest jedynym rodzajem pamięci, jaki człowiek posiada (rys. 1.13).

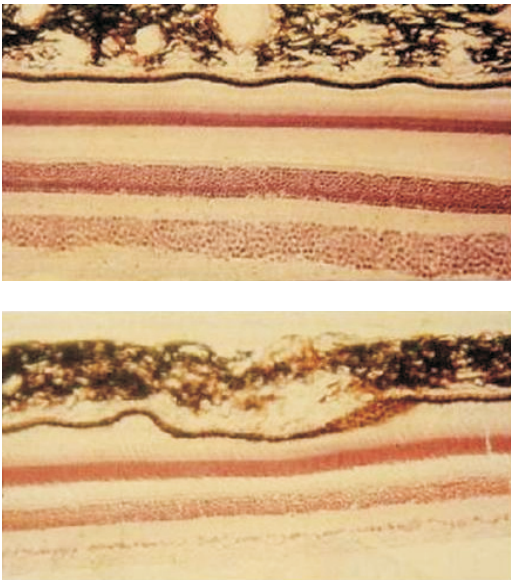
Trzecim źródłem informacji, na którym opierała się koncepcja sieci neuronowych w latach 90., było rozpoznanie w tym czasie wewnętrznej budowy mózgu. Wytrwała praca wielu pokoleń histologów, analiza tysięcy preparatów mikroskopowych i setki mniej lub bardziej udanych prób rekonstruowania trójwymiarowej struktury połączeń elementów nerwowych zaowocowała tym, że dostępne już były wtedy takie schematy, jak schemat przykładowo przedstawiony na rysunku 1.14 (przedstawiający zresztą najmniej efektowny fragment Ośrodkowego Układu Nerwowego, a mianowicie rdzeń kręgowy).

## 1.4. Skąd się wzięła warstwowa struktura sieci neuronowych?

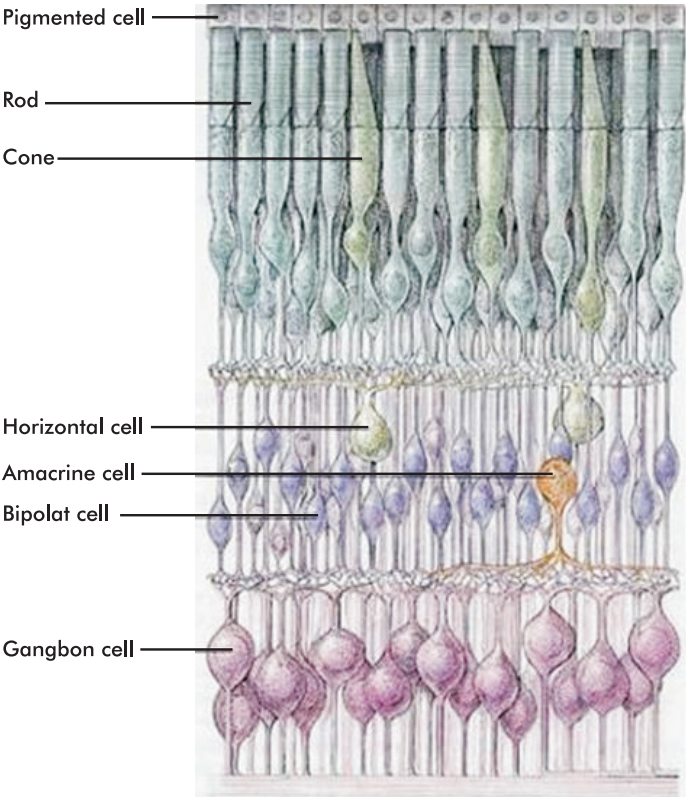
Podobnie jak w przypadku wcześniej wymienionych informacji biologicznych, dotyczących innych właściwości rzeczywistego biologicznego mózgu, także i w odniesieniu do struktury przestrzennej połączeń neuronów przy budowie sztucznych sieci neuronowych całą dostępną wiedzę neuroanatomiczną i cytologiczną maksymalnie uproszczono. Twórcy sieci byli w tym momencie szczególnie radykalni i bezwzględni w swoich działaniach, stosując do budowy i formowania struktur sztucznych sieci neuronowych taką regułę, która była najwygodniejsza w praktyce, chociaż skrajnie uproszczona. Stwierdzono mianowicie, że w niektórych fragmentach kory mózgowej neurony są ułożone w formie regularnych warstw. Wprawdzie nie wszędzie tak jest, ale kilka przykładów takiej właśnie warstwowej struktury da się wyróżnić (rys. 1.15).

Warstwową budowę ma także siatkówka oka (rys. 1.16), będąca – jak wynika z embriologii – przekształconym fragmentem kory mózgowej.

Można więc uznać, że tworząc sieci neuronowe w formie struktur wielowarstwowych dogadzamy wprawdzie swemu wygodnictwu (takie struktury – powtórzmy to jeszcze raz – są z technicznego punktu widzenia najłatwiejsze), nie popełniamy jednak jakiegoś kardynalnego błędu patrząc na sprawę z biologicznego punktu widzenia. Mamy wobec tego prawo sądzić, że wprawdzie sieci neuronowe są krańcowo uproszczonymi modelami fragmen-



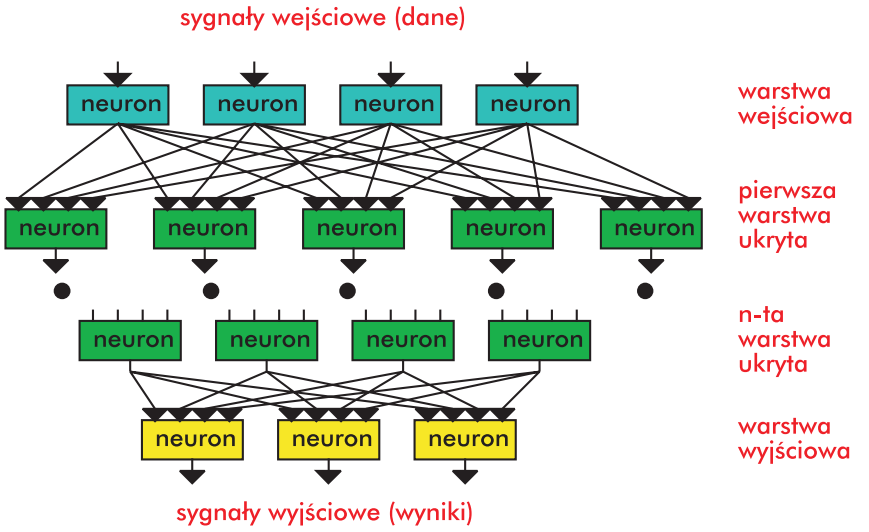
Rys. 1.15. Warstwowa budowa kory mózgowej widoczna na przekrojach fragmentów pola wzrokowego mózgu człowieka




Rys. 1.16. Schemat budowy siatkówki oka ujawnia także jej warstwową budowę

tów rzeczywistej tkanki nerwowej – nie są jednak tak dalece zniekształcone, by wyniki obserwacji uzyskiwanych z ich pomocą nie mogły być poprawnie interpretowane w kontekście potrzeb neurofizjologii.

Typowa sieć neuronowa ma więc strukturę pokazaną na rysunku 1.17.

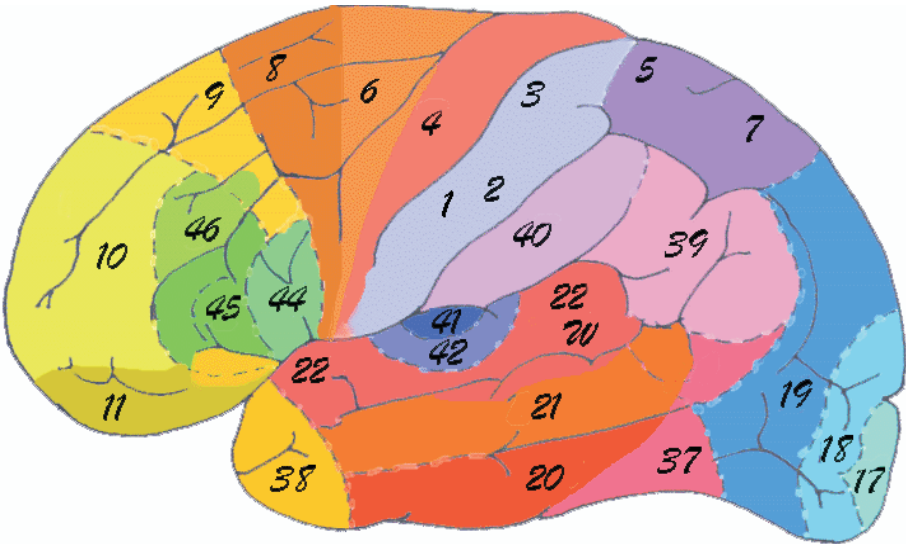



Rys. 1.17. Schematyczna budowa wielowarstwowej sieci neuronowej

Właśnie taka (warstwowa) struktura sieci wyjątkowo łatwo i wygodnie da się wytwarzać zarówno w formie modelu elektronicznego, jak i da się symulować w formie programu komputerowego. Dlatego badacze przyjęli właśnie strukturę warstwową i od tej pory stosują ją we wszystkich sztucznych sieciach neuronowych. Z pełną wiernością biologicznemu oryginałowi ma to niewiele wspólnego, ale jest praktyczne i wygodne. W związku z tym wszyscy tak postępują, nie martwiąc się ani przesłankami biologicznymi, ani dowodami wskazującymi, że architektura sieci bardziej wymyślnie dostosowanej do charakteru zadania może znacznie lepiej realizować stawiane zadania.

Jest jeszcze problem, jak łączyć ze sobą ułożone w warstwy neurony. W rzeczywistym mózgu schemat połączeń jest dosyć skomplikowany i w każdym rejonie mózgu inny. Na podstawie wyróżniania różnych schematów połączeń neuronów została w XIX wieku zbudowana pierwsza topologiczna mapa mózgu, wyróżniająca jego regiony o jednakowych schematach wewnętrznych połączeń między neuronami (rys. 1.18).

Na tej mapie jednakowym kolorem oznaczono te fragmenty o jednakowych (zbadanych mikroskopowo) schematach połączeń, zaś różne kolory odpowiadają obszarom, w których te schematy istotnie się różnią. Ta mapa, zwana planem pól Brodmana, ma dzisiaj raczej historyczne znaczenie, bo obecnie




Rys. 1.18. Cytoarchitektoniczna mapa mózgu według Brodmana. Opis w tekście

potrafimy struktury mózgu różnicować znacznie subtelniej. Jednak sam fakt istnienia mapy Brodmana wskazuje na to, że nie ma prostej i jednoznacznej odpowiedzi na pytanie: *jak są ze sobą połączone neurony w sieci?*, bo odpowiedź zależy od tego, jaki fragment mózgu weźmiemy pod uwagę.

Z kolei biorąc pod uwagę cel budowy sztucznych sieci neuronowych, można byłoby postulować, aby schemat połączeń neuronów w sieci był dostosowany do właściwości rozwiązywanego zadania. Byłby to postulat słuszny, gdyż mamy dowody na to, że dobrze dobrana struktura sieci może znacząco przyspieszyć proces jej uczenia. Problem jednak polega na tym, że większość zadań, które rozwiązujemy z pomocą sieci neuronowych, cechuje się tym, że w nich bardzo niewiele da się powiedzieć z góry o sposobie ich rozwiązywania. Skoro nieznany jest (nawet w przybliżeniu) algorytm, który sieć ostatecznie wybierze po procesie uczenia, to tym bardziej nie wiadomo a priori, jakie elementy w sieci okażą się docelowo potrzebne, a które będą zbyteczne.

Decyzję o sposobie łączenia elementów sieci podejmuje się więc **arbitralnie** i ta arbitralna decyzja jest zwykle taka, że przyjmowany jest schemat połączeń typu „każdy z każdym". Taki pomysł jednorodnego i pełnego schematu połączeń między elementami poszczególnych warstw zapewnia minimalizację wysiłku związanego z definiowaniem sieci, chociaż odbywa się to kosztem dużego nakładu (np. pamięci w modelu symulacyjnym czy potencjalnych połączeń w strukturze chipu), niezbędnego dla zapewnienia możliwości odwzorowania w sieci wszystkich niezbędnych przy takim założeniu połączeń.

Warto zauważyć, że bez przyjęcia takiego założenia o jednorodności struktury sieci samo tylko opisanie topologii jej połączeń wymagałoby podania setek tysięcy parametrów, co skutecznie udaremniałoby wszelkie próby stosowania takich struktur o nadmiernie wymyślnych architekturach połączeń. Natomiast decyzja o przyjęciu połączeń typu „każdy z każdym" stanowi elementarny i nie wymagający żadnego głębszego zastanowienia akt woli konstruktora sieci. Tak jest łatwiej i dlatego (prawie) wszyscy tak robią. Możliwość wygodnego (czytaj: bezmyślnego) stosowania tego typu uproszczenia wynika z faktu, że w trakcie procesu uczenia sieć „sama" wybierze z dostępnych jej początkowo wszystkich połączeń – te, które są naprawdę potrzebne, odrzucając (zerując) pozostałe.

## 1.5. Na ile pierwsze sieci neuronowe były podobne do biologicznego mózgu?

Podsumowując tę część rozważań, trzeba stwierdzić, co następuje: Sztuczne sieci neuronowe, które powstały w latach 90. XX wieku, zostały oparte na tej wiedzy anatomicznej, fizjologicznej i biochemicznej, jaką w tamtym okresie posiadano na temat ludzkiego mózgu. Jednak twórcy sieci neuronowych tę wiedzę potraktowali jako źródło inspiracji, a nie jako wzorzec do dokładnego skopiowania. Dlatego budowa i zasady działania sztucznych sieci neuronowych stosowanych w praktyce **nie są** wiernym odzwierciedleniem stanu wiedzy biologicznej, nawet tamtej, nieco już nieaktualnej, sprzed kilkunastu lat. W rzeczywistości można powiedzieć, że warsztat twórcy nowoczesnych sieci neuronowych składa się w pewnym stopniu z elementów wiedzy biologicznej, które jednak gdzieś tam się unoszą dosyć swobodnie w tle i są raczej źródłem inspiracji, a nie dokładnym wzorcem, na którym opiera on swoje działania. Natomiast materia, którą formuje neurocybernetyk, jest na wskroś informatyczna, gdyż sieci neuronowe powstają, uczą się, są badane i są stosowane w typowych współczesnych komputerach. Graficzną metaforą tego procesu może być zmontowany przez studentów AGH wizerunek autora tej książki, przedstawiający go właśnie jako badacza sieci neuronowych (rys. 1.19).

Dlatego we wszystkich dalszych rozważaniach będziemy pamiętali (i brali pod uwagę), że przy formowaniu sieci neuronowych wszystkie posiadane wiadomości biologiczne najpierw zostały bardzo mocno przetrzebione i uproszczone, a dopiero potem uczyniono z nich podstawę do budowy sztucznych systemów neurocybernetycznych. Fakt ten zasadniczo wpływa na właściwości badanych sieci, które w efekcie są neuronowe bardziej z nazwy niż ze względu na rzeczywiste podobieństwo do prawdziwego mózgu. Wbrew pozorom fakt ten decyduje **pozytywnie** o właściwościach i możliwościach sieci


Rys.1.19. Wiedza o biologicznych neuronach „wsiąkająca" w struktury aktualnie wykorzystywanych komputerów (wizerunek autora książki sporządzony przez studentów AGH)

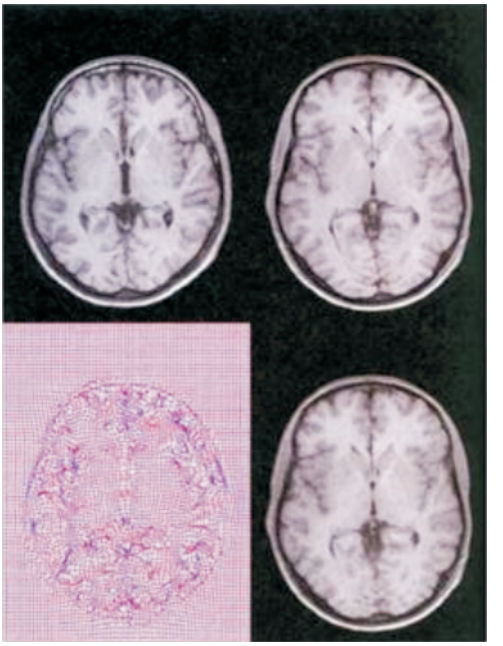
neuronowych – także rozpatrywanych jako narzędzia wspomagające proces biologicznego rozumienia naszego własnego mózgu. Zajmiemy się tym w kolejnym podrozdziale książki.

Przygotowując to kolejne wydanie książki, które ukaże się drukiem ponad dziesięć lat po pierwszym wydaniu, nie mogłem zignorować faktu, że przez tę dekadę badania mózgu poszły znacznie do przodu, a i poglądy na temat tego, kiedy i do czego należy stosować sieci neuronowe, także uległy pewnej ewolucji. W związku z tym w kolejnym podrozdziale, którego nie było w pierwszym wydaniu, porozmawiamy o współczesnych problemach badania systemu nerwowego człowieka oraz spróbuję Ci wskazać, jaką rolę mogą obecnie odegrać sieci neuronowe w tych badaniach.

## 1.6. Jakimi metodami badamy obecnie mózg?

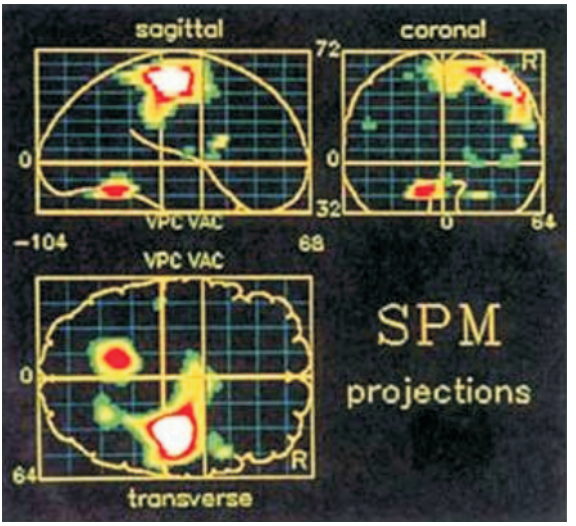
Współczesne instrumentarium, jakim może się posłużyć badacz ludzkiego mózgu w XXI wieku, jest nieporównanie bogatsze niż to, które mieli do dyspozycji odkrywcy, którzy tworzyli te zręby naszej wiedzy o strukturze i działaniu systemu nerwowego, jakie stały się podstawą do stworzenia pierwszych sieci neuronowych w latach 90. ubiegłego stulecia. W pierwszej kolejności trzeba tu wskazać na postęp, jaki za sprawą tomografii komputerowej (CT) oraz magnetycznego rezonansu jądrowego (NMR) nastąpił w zakresie możliwości udoskonalonego obrazowania wewnętrznych struktur mózgu. Dawniej skorupa czaszki, całkowicie nieprzenikliwa dla prymitywniejszych technik

obserwacji z klasyczną techniką rentgenowską włącznie, zazdrośnie kryła tajemnicę wewnętrznej budowy mózgu żywego i zdrowego człowieka. Mózg stawał się dostępny do badań dopiero wtedy, gdy pacjent już zmarł i preparat jego martwego mózgu mógł być poddawany badaniom prowadzonym przez anatomów. Wtedy badacze, którzy kroili go na plasterki, oglądali pod mikroskopem, próbowali prześledzić i zrozumieć strukturę jąder przetwarzających informację oraz schemat ich połączeń – mniej więcej z takim samym powodzeniem, jakby ktoś kroił na plasterki komputer, oglądał na poszczególnych przekrojach jak i którędy przebiegają przewody i układy scalone – i na tej podstawie usiłował zrozumieć zasady informatyki. Dzisiejsze techniki pozwalają swobodnie zajrzeć do mózgu żyjącego człowieka i pokazać jego budowę z zadziwiającą wręcz dokładnością – patrz na przykład rysunek 1.20.



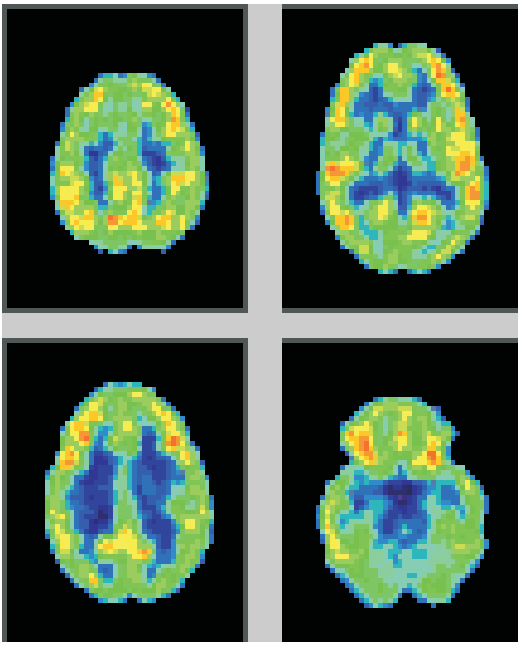
Rys. 1.20. Struktury wewnętrzne mózgu żyjącego człowieka mogą być dziś przedstawiane z zadziwiającą wręcz dokładnością i precyzją

Współczesna technika diagnostyczna pozwala także wykrywać i lokalizować te miejsca, w których aktywność mózgu jest w danym momencie szczególnie duża – patrz rysunek 1.21. Wiążąc takie centra aktywności z rodzajem czynności, jaką w danym momencie wykonuje badana osoba, można dowiedzieć się, które struktury mózgu są zaangażowane w realizację których zdań, a zatem można także lepiej zrozumieć funkcjonalne aspekty poszczególnych struktur nerwowych. Pokażę Ci to na przykładzie.




Rys. 1.21. Metoda prezentacji pokazująca, które regiony mózgu są aktywne przy wykonywaniu określonych czynności

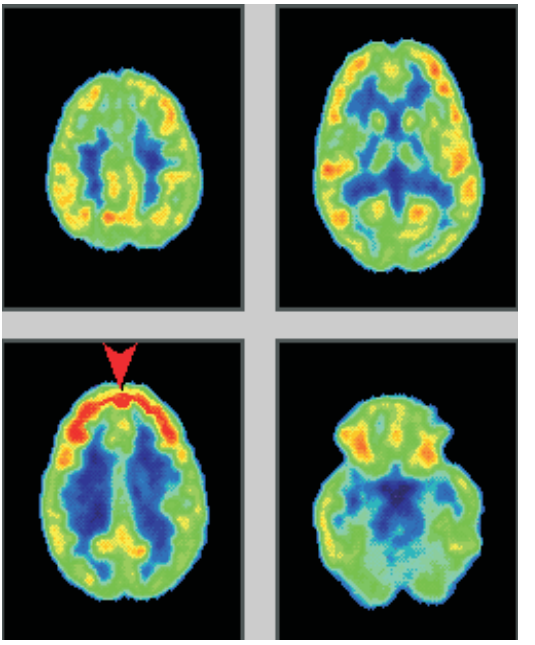
Popatrz na rysunek 1.22. Pokazałem tam (w postaci czterech przekrojów wykonanych na różnej wysokości) uzyskany za pomocą metody badawczej zwanej PET obraz wnętrza mózgu czuwającego człowieka, który w danym momencie nie skupia się na żadnej szczególnej czynności. Widać, że więk-



Rys. 1.22. Obraz mózgu czuwającego człowieka, który w danym momencie nie skupia się na żadnej szczególnej czynności

sza część jego mózgu jest bezczynna (te obszary przedstawione są w kolorze niebieskim albo zielonym), natomiast gdzieniegdzie trwa praca neuronów (to te żółte i czerwone obszary), bo ten człowiek, chociaż zrelaksowany, jednak o czymś myśli, porusza się, doznaje jakichś wrażeń itd. Jednak aktywność ta jest rozproszona i mało intensywna.

Jeśli jednak badany człowiek zacznie aktywnie używać swojego mózgu, na przykład rozwiązując trudne zadanie matematyczne, to obszary odpowiedzialne za abstrakcyjne myślenie natychmiast zwiększą swoją aktywność, co na obrazie pokaże się jako ich intensywne czerwone zabarwienie (rys. 1.23).

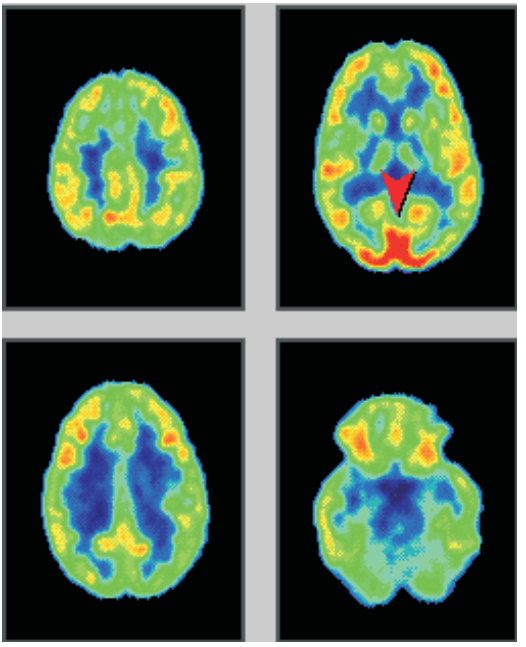


Rys. 1.23. Obraz aktywności mózgu człowieka rozwiązującego trudne zadanie matematyczne. Silne czerwone zabarwienie płatów czołowych (pokazanych strzałką) wskazuje na ich aktywizację

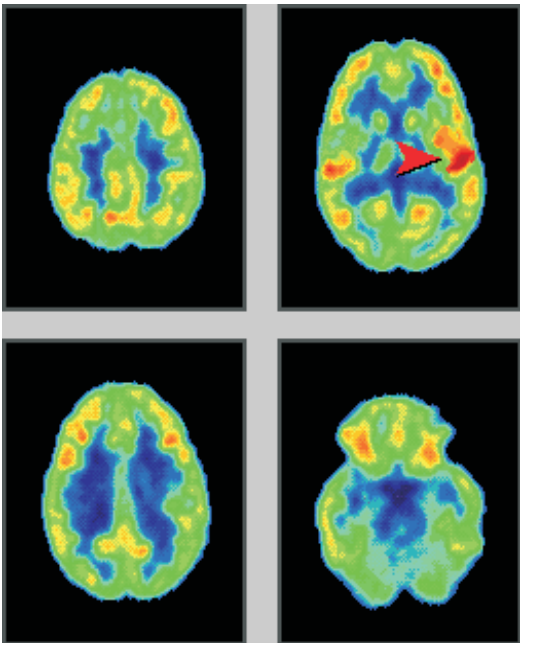
Nie tylko świadomy wysiłek zmuszanego do pracy mózgu można zarejestrować i przedstawić tą metodą. Jeśli badany człowiek zainteresuje się jakimś obrazem – to w tylnych częściach jego mózgu uruchomiona zostanie ogromna liczba neuronów związanych z czynnościami analizy, przetwarzania i rozpoznawania sygnałów wizyjnych (rejestrowanych za pomocą oczu) – rys. 1.24.

Z kolei podczas słuchania i rozumienia mowy aktywizacji podlegają ośrodki związane z rejestracją i analizą dźwięków, zlokalizowane w płatach skroniowych (rys. 1.25).

Opisane wyżej sposoby prezentacji informacji o budowie i o działaniu mózgu mogą rejestrować jego budowę i stan w pewnym określonym momencie, ale pozwalają także śledzić zachodzące w mózgu procesy. W szczególności

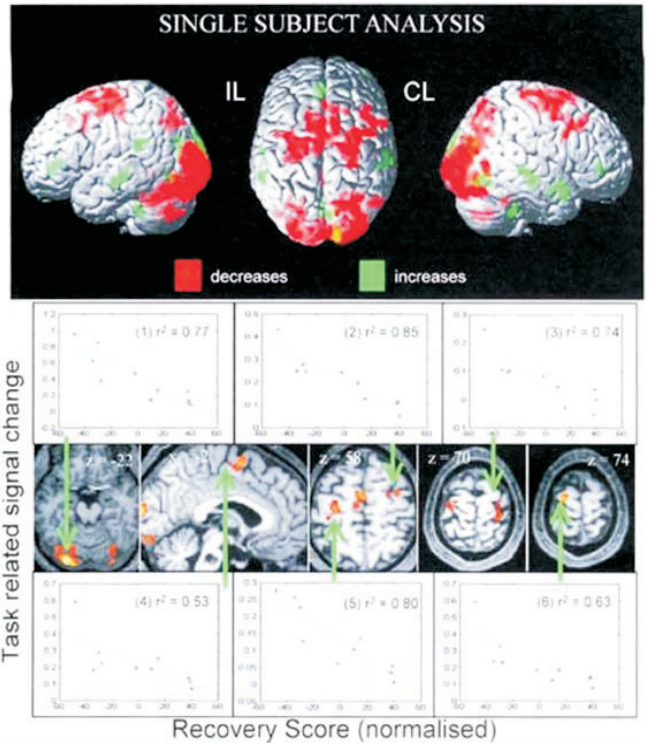


Rys. 1.24. Aktywność mózgu człowieka oglądającego interesującą scenę. Widać aktywizację płatów potylicznych (wskazane strzałką), które są odpowiedzialne za odbiór i rozpoznawanie informacji wizyjnych



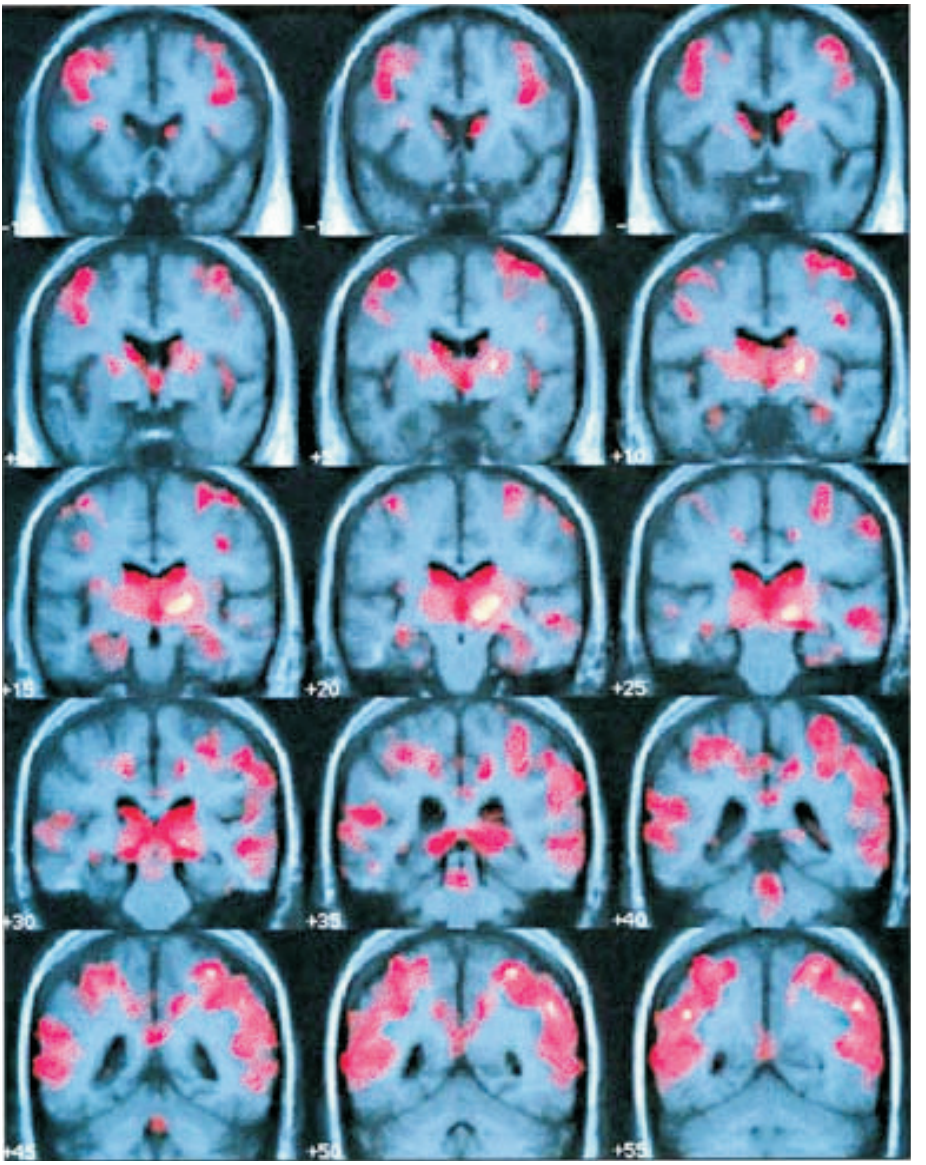
Rys. 1.25. Człowiek słuchający rozmowy aktywizuje głównie płaty skroniowe, bo tam znajdują się ośrodki odbioru i analizy dźwięków. Warto zauważyć, że aktywne są obszary skroniowe tylko po jednej stronie, bo funkcje rozpoznawania mowy umieszczone są w mózgu tylko w jednej półkuli. Gdyby ten człowiek słuchał muzyki, to „zapaliłyby się" skroniowe płaty mózgu symetrycznie po obydwu stronach

dzięki **połączonym** technikom udoskonalonego obrazowania wewnętrznych struktur nerwowych oraz śledzenia aktywności poszczególnych fragmentów tego systemu – można dzisiaj uzyskiwać z wnętrza mózgu wyjątkowo precyzyjne obrazy ilustrujące, które fragmenty kory mózgowej wykazują w określonych sytuacjach wzrastającą, a które malejącą aktywność (rys. 1.26).




Rys. 1.26. Śledzenie zmian w mózgu związanych z wykonywaniem określonych czynności

Co więcej, opisane wyżej zmiany można rejestrować i odtwarzać w sposób **dynamiczny** (jak na filmie lub podczas animacji komputerowej). W książce nie mogę pokazać wspomnianych obrazów w postaci ruchomej, ale mogę zaprezentować wyniki takich dynamicznych badań w podobnej formie, jak pokazuje się czasem poszczególne fazy ruchu sportowca lub kolejne klatki animowanego filmu. Na rysunku 1.27 na kolejnych obrazkach widoczne jest funkcjonowanie określonych struktur anatomicznych wewnątrz mózgu w kolejnych chwilach czasu. Prezentując takie obrazy kolejno z odpowiednią szybkością uzyskuje się – podobnie jak na filmie lub w telewizji) – wrażenie płynnie wykonywanego ruchu, dzięki czemu badacz może swobodnie śledzić, jakie struktury i w jakiej kolejności podlegają aktywizacji przy wykonywaniu poszczególnych analizowanych czynności.




Rys. 1.27. Prezentacja dynamiki procesów zachodzących wewnątrz mózgu

## 1.7. Czy sieci neuronowe mogą pomóc w poznaniu tajemnic ludzkiego umysłu?

Przedstawione wyżej **przykłady** wyników uzyskiwanych za pomocą współczesnych metod badania mózgu zdają się wskazywać na to, że po upływie dekady lat 90. XX wieku, która była zresztą nazwana „dekadą mózgu" ze względu na intensywność odpowiednich badań, wiemy już o ludzkim umyśle tak wiele, że nie ma on dla nas żadnych tajemnic.

Nic bardziej mylnego. Strukturę mózgu poznaliśmy rzeczywiście dosyć szczegółowo, o jego działaniach wiemy też sporo, ale pojawił się problem, który bardzo często występuje w przypadku badania szczególnie skomplikowanych obiektów: rekonstrukcji obrazu całości na podstawie ogromnej liczby oddzielnych, rozproszonych wyników szczegółowych. Jedną z bardzo skutecznych metod, jaką posługuje się współczesna nauka, jest metoda dekompozycji, albo mówiąc prościej – dzielenia na kawałki. Mamy trudność z opisaniem dużego systemu? No to podzielmy go na sto części i zbadajmy każdą z nich z osobna! Nie potrafimy zrozumieć skomplikowanego procesu? To znajdźmy kilkadziesiąt prostych podprocesów, składających się razem na ten skomplikowany, nie poddający się opisowi – i prześledźmy każdy z nich z osobna. Ta metoda jest bardzo skuteczna, bo wyodrębnione w ramach podziału podsystemy czy podprocesy można zawsze ponownie podzielić na mniejsze i prostsze, jeśli się będą opierały naszym metodom naukowym – ale pojawia się przy tym jedna charakterystyczna trudność: Kto, kiedy i w jaki sposób zbierze razem wyniki licznych badań tych wszystkich części i fragmentów?

Przy analizie naprawdę skomplikowanych systemów synteza wyników badań częściowych nie jest sprawą prostą ani łatwą, zwłaszcza gdy te wyniki składowe powstały z użyciem różnych technik badawczych stosowanych w obszarze różnych nauk szczegółowych. Wtedy trudno znaleźć kogoś, kto byłby w stanie scalić na przykład oparte głównie na rysunkach informacje anatomiczne, bazujące na opisach zdarzeń dane fizjologiczne oraz wyniki oznaczeń biochemicznych, uzyskane z aparatury analitycznej. Dlatego tam gdzie tylko można – warto posłużyć się modelowaniem komputerowym, które może pełnić rolę swoistego „wspólnego mianownika" dla tych wszystkich informacji czerpanych i zestawianych z różnych źródeł. Doświadczenia potwierdzają, że komputerowy model opisujący anatomię **da się sprzęgnąć** z komputerowym opisem procesów fizjologicznych oraz z komputerowym zapisem reakcji biochemicznych, zatem tą właśnie drogą próbuje się obecnie rekonstruować **całość** wiedzy o różnych systemach biologicznych, badanych do tej pory rozdzielnie przez różnych badaczy.

W odniesieniu do systemu nerwowego, a zwłaszcza w odniesieniu do mózgu człowieka można używać różnych modeli komputerowych, żeby połączyć wyniki wielu różnych badań i uzyskać szansę na zrozumienie **całościowego** funkcjonowania tego niezwykle złożonego systemu. O niektórych z nich będziemy jeszcze w tej książce wspominać w innych miejscach, natomiast w tym wstępnym rozdziale chciałbym stwierdzić, że **najprościej** można próbować opisać naszą wiedzę o mózgu odwołując się właśnie do rozważanych w tej książce sieci neuronowych. Oczywiście mózg człowieka jest bardzo wielki i bardzo skomplikowany, a sieci neuronowe są w stosunku do niego bardzo małe i w dodatku naprawdę **bardzo** uproszczone. Jednak w nauce często tak bywa, że stosując uproszczone modele doświadczalne dochodzimy do odkrycia prawidłowości, które sprawdzają się i potwierdzają także w bardzo wielkiej skali. Na przykład jeśli chemik w małej probówce przeprowadzi pewną reakcję, to mamy prawo sądzić, że taka sama reakcja przebiega w ogromnym oceanie albo we wnętrzu odległej gwiazdy – i zazwyczaj to się potwierdza. Natomiast znacznie łatwiej manipuluje się małą probówką niż wielkim oceanem, że o odległych gwiazdach nie wspomnę. Przy dochodzeniu do prawdy naukowej **uproszczenie** bywa często kluczem do sukcesu.

Dlatego mimo prostoty (graniczącej z prymitywizmem!) sztucznych sieci neuronowych coraz częściej słyszy się o badaniach, w których sieci te są wykorzystywane do tego, żeby z ich pomocą modelować procesy zachodzące w ludzkim mózgu, uzyskując dzięki temu podstawy do jego lepszego zrozumienia. Żeby jednak uświadomić sobie, w jakim stopniu możemy polegać na tym narzędziu – spróbujmy dokonać krótkiego porównania stopnia złożoności budowanych przez nas sztucznych sieci neuronowych i systemów nerwowych ludzi oraz zwierząt.

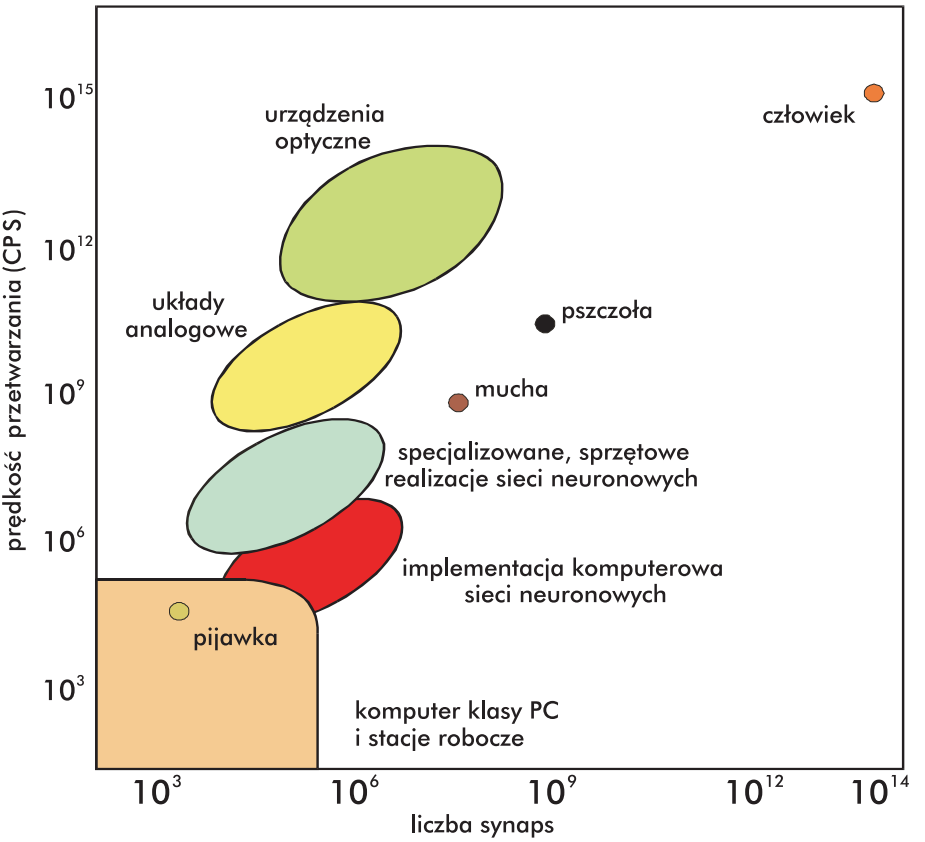
## 1.8. W jakim stopniu sieci neuronowe są uproszczone w stosunku do biologii?

Sieci neuronowe **są** bardzo uproszczone w stosunku do systemów nerwowych większości żywych organizmów. Można to prześledzić na rysunku 1.28, na którym zestawiono właściwości pewnej liczby systemów neurocybernetycznych – zarówno naturalnych, jak i sztucznych. Na rysunku tym oś pozioma określa stopień strukturalnej złożoności określonego systemu (wyrażany przez liczbę występujących w nim synaps, czyli miejsc, w których zarówno biologiczne mózgi, jak i sztuczne sieci neuronowe gromadzą swoją wiedzę). Na osi pionowej z kolei odkładana jest szybkość przetwarzania informacji, osiągalna w danym systemie. Szybkość ta wyrażana jest w specjalnych jednostkach CPS (który to skrót pochodzi od słów *connections per second*); w tym mo-

mencie jednostkom tym nie musimy jednak poświęcać zbyt wiele uwagi. Obie osie wyskalowane są w sposób **logarytmiczny**, to znaczy że nawet niewielkie przesunięcie w górę albo w prawo oznaczać będzie niesłychanie szybki (stu lub tysiąckrotny!) wzrost odpowiedniej wielkości – złożoności mózgu lub jego szybkości. Ten właśnie sposób skalowania wykresu powoduje, że zaznaczono na nim na przykład człowieka, pszczołę i muchę, nie próbowano natomiast zaznaczać mózgu psa albo konia, bo przy logarytmicznym skalowaniu – punkt odpowiadający wielkości i sprawności mózgu psa lokowałby się tak blisko punktu wytyczającego właściwości mózgu człowieka, że obie kropki byłyby nieodróżnialne na wykresie.

Na przedstawionym wykresie widać, że dla wszystkich istot żywych – od pijawki do człowieka – obserwowany jest w praktyce **proporcjonalny** związek zachodzący pomiędzy wielkością mózgu a jego sprawnością. Wynika to z faktu, że biologiczne neurony przetwarzają informacje równocześnie, pracując wszystkie naraz, zatem im jest ich więcej, tym większą pracę mogą wykonać w tym samym czasie. W systemach technicznych nie jest to prawdą, dlatego szybkość ich działania w zasadniczy sposób zależy od tego, w jakiej technologii są one wykonane. Zwykłe urządzenia cyfrowe są w związku z tym wolniejsze od rozwiązań specjalizowanych, a te są wolniejsze od systemów optoelektronicznych. Typowe sieci neuronowe wykorzystywane w technice tworzone są najczęściej w oparciu o komputery klasy PC oraz stacje robocze, więc jak możesz łatwo zauważyć na wykresie, zajmują one obszar w samym lewym dolnym rogu, czyli odpowiadający najmniejszym stopniom złożoności (są **miliony** razy mniej skomplikowane niż ludzki mózg) oraz bardzo małym szybkościom przetwarzania informacji. Do wielu praktycznych zastosowań to wystarczy, ale do tego, żeby zamodelować tą drogą cały mózg, jest naprawdę **bardzo daleko**.

Jak jednak wspomniałem już wcześniej, naszym celem nie jest (na obecnym etapie rozwoju tych badań) modelowanie mózgu **w całości**. Zadanie, jakie stawiają sobie neurocybernetycy, jest daleko skromniejsze, ale dzięki temu całkowicie realistyczne. Chodzi o to, by korzystając z prostoty oraz ograniczonego rozmiaru sieci neuronowych (łatwych w związku z tym do budowy, do obserwowania i do zrozumienia) sprawdzić, jak to się dzieje, że w systemie wzorowanym na ludzkim mózgu dochodzi do uczenia się od nauczyciela lub do samodzielnego zdobywania wiadomości (samouczenia), następnie formowania i uogólniania wiedzy, kojarzenia odległych faktów, klasyfikowania wrażeń i podejmowania decyzji – słowem do wszystkich tych czynności i procesów, które psycholodzy zdążyli już zidentyfikować jako składniki ludzkiej inteligencji, natomiast biolodzy mózgu nie zdołali jeszcze powiązać z konkretnymi mechanizmami zachodzącymi w dających się wskazać strukturach anatomicznych.




Rys. 1.28. Złożoność i sprawność różnych systemów neurocybernetycznych. Opis w tekście

Jeśli podejmiesz trud poznania i zrozumienia sieci neuronowych, to masz szansę włączyć się do tych badań. Warto to zrobić, bo jest to teren, na którym jest wciąż mnóstwo „białych plam", czekających na swoich Kolumbów. Na powierzchni Ziemi nie ma już nie odkrytych wysp ani nie poznanych ludów nawet w głębi tropikalnych dżungli, do eksploracji dalekich planet potrzeba niezwykle kosztownego wyposażenia, natomiast przebogaty i nie do końca odkryty Wszechświat wnętrza naszego mózgu wciąż czeka na swoich odkrywców, a jest przecież tak blisko – dosłownie w zasięgu naszych rąk. Jeśli więc kiedykolwiek kusiła Cię kariera odkrywcy, a także jeśli kiedykolwiek interesowałeś się swoim własnym mózgiem – to nie muszę Cię zapewne dalej namawiać. Jeśli natomiast jest Ci obojętny Twój własny intelekt i jego zadziwiające siedlisko, jakim jest biologiczny mózg – to może zachęci Cię wiadomość, że sieci neuronowe mogą być bardzo skuteczne jako narzędzia obliczeniowe – i to w rozwiązywaniu takich zadań, z którymi typowe komputery

i typowe programy nie bardzo sobie radzą. O tych atutach sieci neuronowych oraz o tych argumentach skłaniających do tego, by je poznawać i ich używać, była już mowa w przedmowie, ale ponieważ sprawa jest ważna, pozwolę sobie do niej jeszcze raz powrócić i dodać jeszcze kilka dalszych argumentów, które z pewnością warto wziąć pod uwagę.

## 1.9. Jakie są główne zalety sieci neuronowych, a także kto i do czego ich używa?

Jak już wiesz, sieci neuronowe są nowym i pod wieloma względami atrakcyjnym narzędziem informatycznym, wzorowanym na budowie mózgu i z tego powodu zasadniczo odmiennym w swoich założeniach od tych komputerów, którymi posługujemy się na co dzień. Najkrócej mówiąc można stwierdzić, że są one w większości zastosowań narzędziem pozwalającym pokonać wiele barier i ograniczeń, trudnych do pokonania innymi metodami, a jednocześnie wybitnie wygodnym i wydajnym. Dlatego właśnie interesują się nimi zarówno badacze, jak i praktycy. Zalety praktyczne sieci neuronowych polegają na dwóch podstawowych atutach:

⇒ stanowią one wygodną i tanią propozycję wieloprocesorowego systemu o bardzo wielu elementach przetwarzających równolegle dostarczane informacje;

⇒ nie wymagają programowania, tylko wykorzystują proces uczenia.

W dalszych rozdziałach omówię dokładniej obydwie wymienione zalety sieci neuronowych, więc teraz tylko wspomnę o nich skrótowo, pokazując, jakie jest ich znaczenie.

Pierwsza zaleta daje się wykorzystać wtedy, gdy tworzy się sieci neuronowe jako wyspecjalizowane urządzenia, wykorzystywane na przykład do bardzo szybkiego przetwarzania obrazów, rozumienia mowy albo sterowania mobilnym robotem. Istota tej zalety polega na tym, że sieć neuronowa zbudowana z wielu oddzielnych neuronów, wykonanych na przykład jako elementy układu scalonego, może pracować w taki sposób, że wszystkie te neurony działają **równocześnie**, rozwiązując postawione zadanie jak dobrze zgrany kolektyw. Dzięki takiemu „równoległemu" sposobowi pracy całego zbioru zaangażowanych neuronów nie trzeba czekać na wynik ich pracy tak długo, jak w przypadku użycia do tego samego zadania typowego komputera, w którym wszystkie czynności musi wykonać jeden procesor, wykonując je w efekcie kolejno, jedna po drugiej, czyli **szeregowo**. Jeśli tysiące sztucznych neuronów składających się na sieć wykonują przypadające im zadania obliczeniowe **równolegle**, to szybkość pracy sieci neuronowych może docelowo **miliony razy** przewyższać szybkość aktualnie eksploatowanych komputerów.

Kiedyś w przyszłości to może być bardzo ważny argument przemawiający za korzystaniem z sieci neuronowych, jako z wyjątkowo wydajnego narzędzia informatycznego, gdyż jest możliwe, a nawet wysoce prawdopodobne, że ta zasada „zmasowanej równoległości obliczeń" (warto zapamiętać rozpowszechniony w literaturze angielski termin „*massive parallel processing*") znajdzie szerokie zastosowanie. Obecnie jednak bardzo nieliczne są systemy obliczeniowe, zawierające sztuczne neurony jako swoje elektroniczne elementy składowe, dlatego o tej zalecie jedynie wspominam, nie rozwijając tego tematu szerzej. Żałuję przy tym bardzo, że nie mogę Ci zapewnić dostępu do takich specjalistycznych systemów elektronicznych, na przykład tych, dzięki którym w moich laboratoriach badawczych w Akademii Górniczo-Hutniczej (w Krakowie) prowadzimy badania (między innymi) nad wykorzystaniem sieci neuronowych jako wieloprocesorowych, bardzo szybkich układów obliczeniowych – zwłaszcza przetwarzających obrazy w systemach wizyjnych robotów przemysłowych. To naprawdę zmyślne i zabawne maszyny!

Jednak nie uda mi się udostępnić Ci tych urządzeń ani na kartach książki, ani w Internecie (w którym znajdziesz jednak przeznaczone dla Czytelników tej książki ciekawe programy, symulujące działanie sieci neuronowych na dowolnym dostępnym dla Ciebie komputerze, o czym jeszcze dodatkowo powiem Ci kilka słów za chwilę). Nie mogąc Ci pokazać działających specjalizowanych neurokomputerów, pracujących bardzo szybko dzięki równoległej pracy elektronicznych neuronów, nie będę o tym więcej wspominał i w całej dalszej treści tej książki będą skupiał Twoją uwagę głównie na drugim atucie, to znaczy na konsekwencjach możliwości uczenia sieci neuronowych.

A jest to atut ważki i mający daleko idące konsekwencje. Najważniejszym następstwem faktu, że sieć neuronowa może **sama** zdobywać wiedzę poprzez proces uczenia się, jest możliwość uwolnienia użytkownika tych systemów od mozolnego i czasochłonnego programowania. Nawet jeśli ktoś bardzo lubi programować komputery (a większość użytkowników informatyki **nie** lubi, czego dowodem jest powodzenie, jakim się cieszą gotowe programy pisane przez innych), to musi przyznać, że **szybciej i wygodniej** można rozwiązać problem, jeśli programu nie trzeba pisać. Dla rozwiązania problemu za pomocą sieci nie potrzeba pisać programu, gdyż sieć się sama nauczy, jak go trzeba rozwiązywać. Wystarczy, że potrafię pokazać sieci przykłady zadań wraz z ich prawidłowymi rozwiązaniami – a metodę i regułę rozwiązywania sieć stworzy sobie sama. To duży zysk na czasie i duża wygoda.

Nie jest to jednak jedyna zaleta. Skoro sieć może się sama nauczyć rozwiązywania różnych trudnych zadań, dla których ja wcale nie muszę pisać programów (chociaż mógłbym, bo wiem, jak je należy rozwiązywać) – to może tę samą metodę uczenia da się zastosować dla takich zadań, dla których

**ja sam nie znam zasad, według których można by je rozwiązać**, ale za to mam do dyspozycji przykłady takich zadań wraz z rozwiązaniami i mogę ich użyć do uczenia sieci. Innymi słowy, sieć neuronowa dzięki uczeniu się może poszerzyć krąg rozwiązywalnych zadań o te zadania, których ja sam bym nie umiał rozwiązać, ale dla których pewne gotowe rozwiązania istnieją. Zauważ, jak ogromnie poszerza to krąg potencjalnych zastosowań informatyki. Dzięki uczeniu sieci neuronowych jestem w stanie włączyć w obszar osiągalnych technicznie celów niesłychanie szeroki zbiór zadań, charakteryzujących się tym, że potrafię dla nich jasno sprecyzować cele i pokazać przykłady praktycznego ich osiągania, ale nie potrafię nic powiedzieć na temat sposobów i metod osiągania tych celów. We wstępie do rosyjskiego wydania tej książki (powtórnie namawiam Cię, żebyś go przeczytał, jeśli jeszcze tego nie zrobiłeś) pisałem, że dotyczy to w pierwszej kolejności takich umiejętności, które człowiek posiada, nie mając świadomości tego, jak właściwie uzyskuje on swoje sukcesy. Jako przykład takich umiejętności wskazałem wtedy rozpoznawanie ludzi na podstawie obrazów ich twarzy. Człowiek robi to bardzo łatwo i bez zastanowienia, ale kiedy wprowadzi się fotografię ludzkiej twarzy do komputera, to absolutnie nie wiadomo, jak analizować i przetwarzać informacje o ciemniejszych i jaśniejszych pikselach składających się na ten obraz, by ustalić tożsamość fotografowanej osoby – niezależnie od kąta widzenia kamery, położenia twarzy, oświetlenia kadru itp.

Tutaj chciałbym Ci dodać sugestię, że ta metoda uczenia sieci neuronowych może być także użyta do rozwiązywania takich zadań, co do których można mieć wątpliwości, czy w ogóle **istnieje** jakaś metoda ich rozwiązywania. Jako przykład tej klasy zadań mogę wskazać zagadnienia prognozowania różnych przyszłych zdarzeń na podstawie zdarzeń, które miały miejsce w przeszłości. Takie prognozy są ważne i potrzebne. Można wiele zyskać, jeśli potrafi się trafnie przewidzieć, co się zdarzy w przyszłości. Potrzebujemy prognoz gospodarczych (na przykład pozwalających określić, które akcje na giełdzie będą drożały, a które taniały). Potrzebujemy skuteczniejszych metod przewidywania pogody. Potrzebujemy prognoz demograficznych. Chcemy przewidywać powodzie, trzęsienia ziemi i inne kataklizmy. Przykładów potrzebnych nam prognoz jest bez liku.

Zapewne łatwo zgodzisz się ze mną, gdy powiem, że we wszystkich przypadkach, kiedy potrzebna jest nam prognoza, musimy korzystać z założenia, że w **przeszłych** zdarzeniach zawarte są przesłanki do przewidywania **przyszłości.** Jednak ani Ty, ani ja, ani nikt inny nie zna ścisłych reguł prognozowania przyszłości na podstawie przeszłości. No, może poza najprostszymi przypadkami całkowicie deterministycznych procesów, takich jak jednostajny ruch planet (łatwo można przewidzieć, gdzie będzie planeta nawet za tysiąc

lat) oraz poza raczej prostymi przypadkami, w których są znane zależności przyczynowo-skutkowe. Na przykład wiadomo, że gdy postawisz wodę w garnku na silnym ogniu, to się zagotuje. W problemach, dla których założenie o determinizmie nie do końca da się udowodnić (na przykład w procesach gospodarczych), a także w przypadkach, w których brakuje reguł, które pozwalają wiązać przyczyny ze skutkami – prognozowanie jest bardzo trudne. Tymczasem z literatury wiadomo, że sieci neuronowe wielokrotnie z powodzeniem stosowano do tego, by przewidywać różne takie nie zdeterminowane oraz pozbawione zauważalnej przyczynowości zjawiska i procesy, a trafność prognoz proponowanych przez „sieciowego wróżbitę" zaskakiwała często samych twórców rozważanych sieci neuronowych. Tą drogą udało się uzyskać trafne prognozy giełdowych cen poszczególnych spółek, prognozy kursów walut, prognozy zapotrzebowania na energię elektryczną, prognozy meteorologiczne i hydrologiczne i bardzo wiele innych. Kluczem do sukcesu we wszystkich tych przypadkach był fakt, że dysponowano dużą liczbą danych retrospektywnych, które mogły być wykorzystane do tego, żeby sieci neuronowej pokazać odpowiednie przykłady i w ten sposób nauczyć ją skutecznego przewidywania. Zasada działania jest w tym przypadku bardzo prosta. Mając dane z przeszłości, można traktować je jako przykłady związków między poprzedzającą sekwencją pewnych zjawisk a następującym po nich (również znanym, bo mającym miejsce w historii) interesującym końcowym faktem. Na przykład, mając do dyspozycji informacje z poprzednich lat mogę łatwo skompletować zestaw danych, zaczynający się od zbioru obserwacji z – przykładowo – poprzedzających jakiś interesujący moment czasu dwóch tygodni oraz podający dokładnie, co się po tych dwóch tygodniach stało. Jeśli takich danych zbierze się dostatecznie dużo – wówczas można sieć nauczyć, by mając obserwacje z przeszłości nauczyła się przewidywać przyszłość – niezależnie od tego, jaki charakter ma związek między przeszłymi danymi i przyszłością. Zbudowano już takich prognostycznych sieci całe mnóstwo – i wiele z nich świetnie działało!

W cytowanej wcześniej mojej książce[1] wymieniłem i omówiłem kilkanaście ich zastosowań, tutaj przedstawię kilka dalszych, nowych przykładów, eksponując głównie te dokonania, które znalazły konkretne miejsce w praktyce, zwłaszcza te najnowsze:

⇒ NASA wykorzystuje sieci neuronowe do sterowania ramieniem robota, którego zadaniem jest chwytanie przedmiotów znajdujących się w dowolnym położeniu (dotyczy to między innymi manipulatora działającego w ładowni promów kosmicznych, gdzie w warunkach nieważkości szczególnie trudno

[1] R. Tadeusiewicz: *Sieci neuronowe*, AOW, wyd. III, Warszawa 1993.

jest utrzymać manipulowane obiekty w stałym położeniu), a pewność chwytu jest uzależniona od stopnia zgodności orientacji przestrzennej osi przedmiotu i elementów chwytaka.

⇒ Również do zadania sterowania ramienia robota dedykowana jest sieć neuronowa zbudowana przez uczonych z *New York University Medical Center*. Zastosowano tutaj sieć neuronową jako alternatywę dla złożonych tensorowych obliczeń dotyczących parametrów ruchu ramienia robota w celu znacznego przyspieszenia działania systemu sterowania, który dzięki zastosowaniu sieci może działać w czasie rzeczywistym, podczas gdy zwykłe algorytmy realizowane na tych samych komputerach nie są w stanie osiągnąć wystarczającej szybkości przy wyznaczaniu wszystkich potrzebnych współrzędnych, przyspieszeń i wielkości wymaganych sił i momentów obrotowych.

⇒ Firma *General Dynamics* opracowała dla potrzeb *US Navy* oparty na sieciach neuronowych system klasyfikujący i rozpoznający sygnały sonarowe. System ten pozwala rozpoznawać szumy charakterystyczne dla napędów różnych typów statków i okrętów. Udało się tak dokładnie wytrenować sieć, że jest ona zdolna do rozróżniania między sobą szumów różnych okrętów tego samego typu, co pozwala na identyfikację nazwy jednostki, a także potrafi rozpoznawać za pomocą hydrofonów niektóre dźwięki nadwodne – na przykład szum wytwarzany przez helikopter unoszący się nad powierzchnią oceanu.

⇒ Podobne osiągnięcia, ale w zakresie identyfikacji samolotów uzyskali naukowcy z *University of Pennsylvania* współpracujący z firmą TRW. Zbudowana przez nich sieć neuronowa potrafi rozpoznawać samoloty z taką dokładnością, że podstawą do bezbłędnej identyfikacji może być detal wielkości 18 cali zaobserwowany z odległości 50 mil.

⇒ W szpitalu *Anderson Memorial Hospital* w Południowej Karolinie sieci wykorzystano do realizacji procesu optymalizacji leczenia, zyskując (jak podają publikacje) miliony dolarów oszczędności i ratując życie kilkudziesięciu ludzi. Niestety, bliższe dane na temat tego ciekawego przedsięwzięcia nie są znane.

⇒ Wielki producent rakiet, firma *General Devices Space Systems Division*, użyła sieci neuronowych do sterowania pracą 150 zaworów doprowadzających paliwo i tlen do silników rakiety Atlas. Okazało się, że odpowiednio wytrenowana sieć potrafiła w oparciu o dane na temat aktualnych fluktuacji mocy silnika sterować zaworami trafniej niż poprzednio stosowany, kosztowny i zawodny złożony system automatyki, oparty na algorytmicznym wykorzystaniu danych z setek sensorów.

⇒ Bardzo znanym zastosowaniem sieci neuronowej jest wykorzystany przez firmę *Eaton Corporation* układ sterowania pozwalający na wspomaga-

nie kierowcy dużej ciężarówki (pięć osi, osiemnaście kół) przy wykonywaniu niektórych szczególnie trudnych manewrów (cofanie z naczepą).

⇒ Sieci neuronowe często stosowane są w energetyce – zwłaszcza do prognozowania zapotrzebowania na moc elektryczną. Jedno z bardziej znanych wdrożeń w tym zakresie dotyczy elektrowni *BC Hydro w Vancouver*, szczególnie trudnej do sterowania, ponieważ jej cztery turbiny mają wyraźnie różniące się charakterystyki i zadanie ich włączania i wyłączania w odpowiedzi na zmienne zapotrzebowanie na moc elektryczną ze strony odbiorców energii jest wyjątkowo złożone i niewdzięczne.

Takich i podobnych doniesień o udanych zastosowaniach sieci neuronowych znaleźć można w najnowszym piśmiennictwie bez liku. Żeby nie przedłużać nadmiernie tego wątku, wymienię jeszcze tylko kilka dalszych zastosowań w skrócie.

⇒ Amerykańskie siły powietrzne (US Air Force) stosują sieci neuronowe do rozwoju symulatorów lotu.

⇒ Do sieci neuronowych odwołał się koncern *Ford Motor Company* przygotowując nowy system diagnostyczny dla swoich silników.

⇒ Firma Halliburton stosuje sieci neuronowe do identyfikacji typu skał napotykanych podczas prowadzenia odwiertów poszukiwawczych przy poszukiwaniu złóż ropy i gazu.

⇒ Linie TWA stosują sieci neuronowe do poszukiwania bomb w swoim terminalu na lotnisku JFK w Nowym Yorku.

Potrzebujesz dalszych przykładów? Zajrzyj do najbliższego numeru jakiegokolwiek czasopisma fachowego związanego z dowolną dziedziną wykorzystującą techniki obliczeniowe. Ręczę, że znajdziesz tam jakiś przykład zastosowania sieci neuronowej! Albo wpisz hasło *Neural Networks* do ***Google*** lub do jakiejkolwiek innej wyszukiwarki internetowej. Ręczę, że dostaniesz odnośniki do **milionów** stronic internetowych, na których różni autorzy opisali, do czego i w jaki sposób zastosowali sieci neuronowe – oraz co im z tego wyszło.

Skoro sieci neuronowe stosują obecnie **wszyscy** i **do wszystkiego**, to nie od rzeczy będzie postawić sobie pytanie:

## 1.10. Czy sieci neuronowe wyprą tradycyjne komputery?

Jak widać z zestawienia podanego w poprzednim podrozdziale – udanych przykładów zastosowania sieci neuronowych jest bardzo wiele. Czy z tego wynika, że powinniśmy porzucić dobrze znane, „klasyczne" komputery i wszystkie zadania obliczeniowe rozwiązywać wyłącznie przy użyciu sieci?

Zdecydowanie nie – sieci są narzędziem modnym i wygodnym, mającym jednak szereg bardzo istotnych ograniczeń. Dokładniejsza specyfikacja tych

ograniczeń wyłoni się z dalszych rozważań, kiedy to w kolejnych rozdziałach poznamy kolejne struktury sieci, metody ich uczenia i szczegółowe sposoby ich zastosowania do konkretnych zadań. Już teraz można jednak podać kilka charakterystycznych cech zadań, których rozwiązanie z wykorzystaniem sieci jest **praktycznie niemożliwe.**

Pierwsza klasa zadań, przy rozwiązywaniu których trzeba całkowicie zrezygnować z sieci, związana jest z manipulacjami na symbolach. Wszelkie formy przetwarzania informacji w postaci symbolicznej są niesłychanie trudne do przeprowadzenia przy użyciu sieci neuronowych, dlatego występowanie elementu polegającego na przetwarzaniu symboli powinno być sygnałem, że z użycia sieci trzeba będzie zrezygnować. Sprowadzając rzecz do krańcowego przykładu, można powiedzieć, że nie ma elementarnego sensu budowanie edytora tekstu działającego w oparciu o koncepcję sieci neuronowej lub procesora wyrażeń algebraicznych wykorzystującego tę samą koncepcję.

Drugim „klasycznym" obszarem, w którym z użycia sieci neuronowych trzeba zrezygnować, jest problematyka obliczeń, przy których wymagana jest wysoka dokładność wyniku numerycznego. Sieć pracuje zawsze w sposób „jakościowy", to znaczy dostarczane przez nią rozwiązania mają zawsze charakter przybliżony. Dokładność tego przybliżenia jest zadowalająca w wielu zastosowaniach: przy przetwarzaniu sygnałów, analizie obrazów, rozpoznawaniu mowy, prognozowaniu wartości notowań giełdowych, sterowaniu robotów, aproksymowaniu wartości złożonej funkcji itp. Absolutnie nie do przyjęcia jest jednak przybliżone wykonywanie obliczeń charakterystyczne dla sieci neuronowej w przypadku na przykład obsługi kont bankowych lub realizacji precyzyjnych obliczeń inżynierskich.

Trzeci wreszcie obszar, w którym trudno oczekiwać dobrych wyników przy stosowaniu sieci neuronowych, związany jest z zadaniami wymagającymi wieloetapowego rozumowania – na przykład wieloetapowego rozstrzygania o prawdziwości lub fałszywości pewnych sekwencji stwierdzeń logicznych. Sieć na ogół usiłuje rozwiązać postawiony problem jednokrokowo – jeśli się jej to uda, wynik jest dostępny natychmiast i jest to duży praktyczny sukces. Jeśli jednak trzeba przeprowadzić pewien **ciąg** rozumowania, a zwłaszcza jeśli trzeba podać dokumentację tego ciągu częściowych uzasadnień końcowego wniosku (na przykład w systemach ekspertowych) – sieć okazuje się tworem zupełnie nieprzydatnym i wszelkie próby jej zastosowania prowadzić muszą do frustrujących niepowodzeń.

## 1.11. To może nie warto zajmować się sieciami neuronowymi?

Z tego, co wyżej napisano, nie należy wyciągać pochopnych wniosków: sieć nie może wprawdzie sama wykonywać obliczeń symbolicznych, ale może wspomagać systemy operujące na symbolach w zakresie czynności, z którymi one sobie nie radzą. Przykładem mogą być sieci *Teuvo Kohonena* lub klasyczna sieć NetTalk *Terence Sejnowskiego* używana do zamiany tekstu ortograficznego na ciąg symboli fonematycznych, służących do sterowania syntezatorem mowy. Podobnie ze stwierdzenia, że sieci nie nadają się do obsługi pracy terminala kasjerki w okienku bankowym, nie wynika ich bezużyteczność dla banków jako takich – przeciwnie, okazały się wyjątkowo użyteczne w badaniach wiarygodności kredytobiorców lub przy ustalaniu warunków negocjowanego kontraktu. Podobne zastrzeżenia można mnożyć bez końca. Ważny jest jednak jeden konkretny końcowy wniosek:

**Sieć neuronowa nadaje się do wielu zastosowań, nie jest jednak tak uniwersalna jak klasyczny komputer.** Dlatego entuzjastom łatwo jest wskazać zadania, których rozwiązania za pomocą sieci okazały się znacząco lepsze niż rozwiązania uzyskane przez klasyczne komputery, zaś malkontenci bez trudu wskażą zadania, dla których rozwiązania wykorzystujące sieci okazały się groteskowo niepoprawne. Prawda – jak zwykle – leży pośrodku. Spróbujemy jej poszukać wspólnie w kolejnych rozdziałach, opisując dokładniej technikę sieci neuronowych, metody ich uczenia, a także dołączone do książki bardzo proste programy, które specjalnie przygotowałem po to, by każdy mógł „pomacać własnymi rękami", ile to wszystko jest warte.

W tym rozdziale pokazałem Ci genezę sieci neuronowych oraz opowiedziałem o niektórych ich zaletach. Jedno i drugie może się okazać tym powodem, dla którego uznasz za celowe zapoznanie się z tym narzędziem. Są tacy, dla których urok trafnej prognozy giełdowej przewyższa wszystkie inne zalety sieci. Być może i Ty także tak na to patrzysz. Pozwól jednak, że ja pozostanę przy moich fascynacjach i przy okazji omawiania sieci neuronowych, traktowanych jako narzędzia do różnych mniej czy bardziej użytecznych celów praktycznych, pokażę Ci jednak także, jak bardzo są one ciekawe na skutek tego, że są wzorowane na budowie i zasadach działania żywego mózgu. Opiszę Ci dalej budzące respekt sukcesy, jakie osiągnęli inżynierowie, ekonomiści, geolodzy i lekarze używający sieci do doskonalenia swoich zadań, pokażę Ci, w jaki sposób sieci pomogły im w rozwiązywaniu ich problemów, ale zwrócę Ci także uwagę na konsekwencje, jakie te badania mają dla zrozumienia naszego własnego rozumu. Przekonasz się, jak wiele można wywnioskować z zachowania się sieci jako systemów uczących się i samodzielnie

zdobywających wiedzę. Dzięki dostępowi do moich programów w Internecie (adres masz podany w przedmowie) będziesz nie tylko świadkiem, ale także czynnym **uczestnikiem** zaskakujących doświadczeń, które pozwolą Ci lepiej zrozumieć sieci neuronowe, a przez to być może także lepiej poznać Twój własny mózg, tajemniczy i bogaty w różne zdolności, o którym już Szekspir powiedział, że jest ***…kruchym domem duszy…***

A zatem: czy wiesz już, dlaczego warto zająć się sieciami neuronowymi?

Jeśli tak i jeśli nadal chcesz je poznać, to ta książka pomoże Ci zrealizować Twój zamiar. Zapraszam do lektury dalszych rozdziałów! Tekst książki jest aktualny i wyczerpujący, to znaczy postarałem się zaprezentować w niej wszystko to, co w sieciach neuronowych najważniejsze i najnowsze. Jest on jednak pozbawiony (celowo) tych licznych, drobnych ale uciążliwych szczegółów, w które obfituje każda dziedzina techniki obliczeniowej, a które dają się zwykle wyrazić wyłącznie przy użyciu formuł matematycznych. Potraktowałem bowiem poważnie zasadę zapisaną na okładce wspaniałej książki Stephena W. Hawkinga *Krótka historia czasu,* głoszącą, że każde równanie zamieszczone w tekście zmniejsza liczbę Czytelników o połowę. Zależy mi na tym, by książkę przeczytały liczne osoby, wśród których znajdą się zapewne takie, które po jej przeczytaniu uznają sieci neuronowe za fajne i pożyteczne narzędzie. Dlatego nie zamieszczę w głównym tekście książki ani jednego równania, jeśli bowiem choć jednego Czytelnika może to zniechęcić, to uważałbym to za niepowetowaną stratę. Natomiast każdemu rozdziałowi towarzyszą w oddzielnych ramkach syntetycznie podane najważniejsze informacje teoretyczne, z których możesz skorzystać, jeśli chcesz koniecznie poznać technikę sieci neuronowych od tej mniej barwnej, ale ważniejszej, wysoce zmatematyzowanej strony.

## 1.12. Pytania kontrolne i zadania do samodzielnego wykonania

1. Wymień kilka przykładowych obszarów problemowych, które są modne i chętnie uprawiane we współczesnej informatyce **poza** sieciami neuronowymi.
2. Jakie funkcje i właściwości mózgu zostały najwcześniej poznane i w jaki sposób tę wiedzę zgromadzono?
3. Kto jako pierwszy udowodnił, że mózg składa się z ogromnej sieci połączonych ze sobą, ale jednak anatomicznie i fizjologicznie **odrębnych** neuronów?
4. Jakie zwierzę doświadczalne posłużyło do zdobycia podstawowych informacji na temat elektrochemicznych funkcji mózgu, dzięki czemu Hodgkin

i Huxley opracowali dokładny model tych funkcji i otrzymali za to Nagrodę Nobla?

5. Wymień podstawowe części naturalnego (biologicznego) neuronu.

6. Dlaczego w opisie funkcjonowania systemów neuronowych (biologicznych i sztucznych) takie duże znaczenie odgrywają **synapsy?**

7. Co to są neuromediatory? Czy możesz podać jakieś nazwy i charakterystyki tych pożytecznych związków?

8. Z jakich powodów sztuczne sieci neuronowe mają z reguły budowę warstwową?

9. Jak są łączone ze sobą neurony poszczególnych warstw sieci i dlaczego przyjęto taki właśnie sposób ich łączenia?

10. Jakie są nowoczesne metody badawcze wykorzystywane przy zdobywaniu wiedzy o strukturze i funkcjach mózgu i innych biologicznych struktur neuronowych?

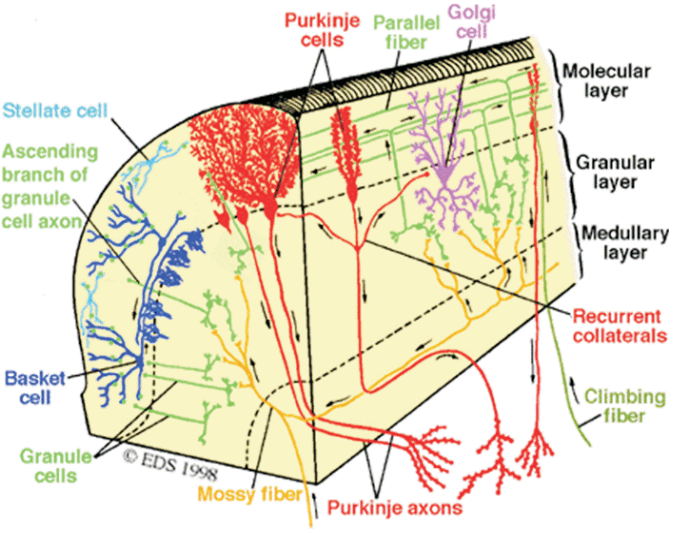
11. Wymień kilka przykładowych obszarów zastosowań sieci neuronowych.

12. Jakie są przewagi sieci neuronowych nad typowymi komputerami i jakie są ich słabe strony i mankamenty?

# 2. Struktura sieci

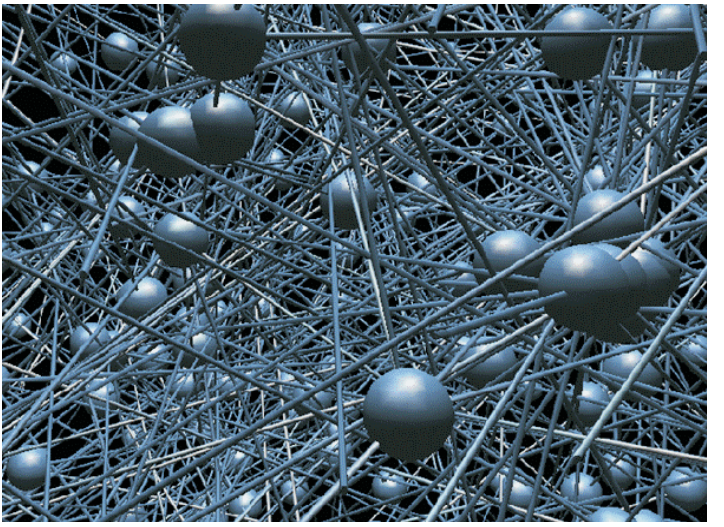
## 2.1. Jak to jest zbudowane?

Któż z nas nie zaczynał poznawania świata od rozkręcenia budzika lub rozłupania magnetofonu – żeby zobaczyć, co jest w środku? Dlatego zanim opowiem Ci o tym, jak sieć działa i jak jej używać – postaram się w miarę dokładnie i w miarę prosto opisać, jak jest ona zbudowana. Jak już wiesz z poprzedniego rozdziału – sieć neuronowa jest systemem dokonującym określonych obliczeń na zasadzie równoczesnej pracy wielu połączonych ze sobą elementów zwanych neuronami. Taka budowa zaobserwowana została pierwotnie w biologicznym systemie nerwowym (na przykład w móżdżku człowieka, którego fragment schematycznie przedstawiłem na rysunku 2.1).




Rys. 2.1. Schematyczny obraz kory móżdżku pokazuje, że biologiczny system nerwowy zbudowany jest z wielu połączonych ze sobą neuronów. Ta sama zasada obowiązuje w odniesieniu do sztucznych sieci neuronowych

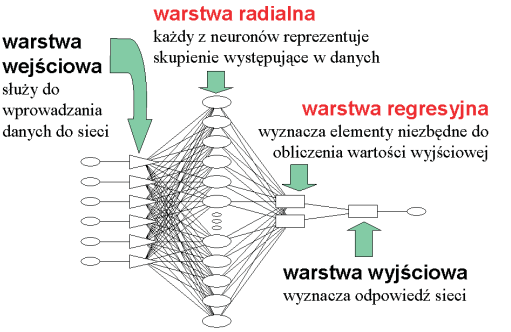
Sieci neuronowe są także zbudowane z wielu neuronów, tyle tylko że sztucznych, to znaczy znacznie uproszczonych w stosunku do oryginału, a także znacznie prościej (by nie powiedzieć – prymitywniej) ze sobą połączonych. Gdyby spróbować zbudować sztuczną sieć neuronową wzorując się na strukturze rzeczywistego systemu nerwowego, to powstały twór byłby bardzo mało przejrzysty i w dodatku trudno by się go kontrolowało. Na rysunku 2.2 pokazałem w przybliżeniu, jak wyglądałaby sieć sztucznych neuronów zbudowana według tych samych schematów strukturalnych, jakie można odnaleźć w rzeczywistym systemie nerwowym. Łatwo zauważysz, że taka struktura nie jest przyjazna do prowadzenia z jej użyciem jakichkolwiek eksperymentów. Przeciwnie – można się w niej zgubić jak w lesie!



Rys. 2.2. Sztuczna sieć o strukturze wzorowanej na trójwymiarowej mapie mózgu byłaby niewygodna w użyciu

Dlatego sztuczne sieci neuronowe budujemy tak, żeby ich struktura była wygodna do prześledzenia oraz tania w realizacji. W efekcie są one po pierwsze **płaskie** (a nie trójwymiarowe) i mają narzuconą regularną budowę, w której występują warstwy neuronów o dobrze zdefiniowanym przeznaczeniu, łączone według prostej, chociaż rozrzutnej zasady połączeń typu „każdy z każdym” (porównaj rysunek 2.3).

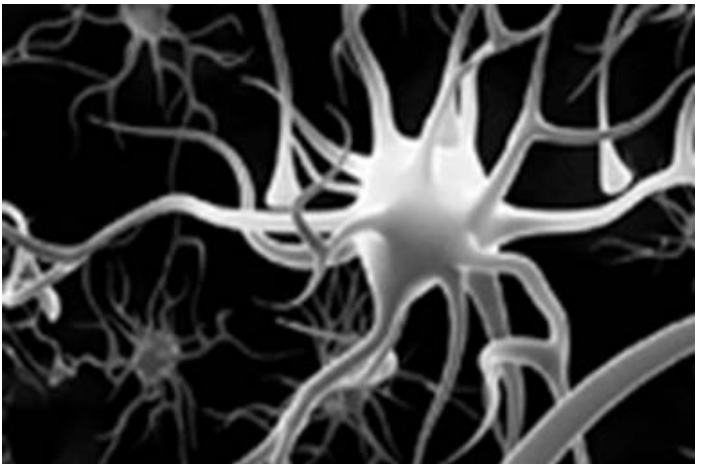
O właściwościach i możliwościach sieci neuronowych decydują trzy czynniki: a) to, z jakich elementów sieć jest zbudowana (czyli jak wyglądają oraz jak działają sztuczne neurony), b) jak te elementy są ze sobą połączone oraz c) w jaki sposób ustala się parametry w sieci za pomocą procesu uczenia. Zajmiemy się teraz tymi czynnikami po kolei.




Rys. 2.3. Schemat praktycznie używanej sieci neuronowej (typu GRNN) pokazuje, że jej budowa jest uproszczona w stosunku do biologicznego oryginału i silnie zracjonalizowana

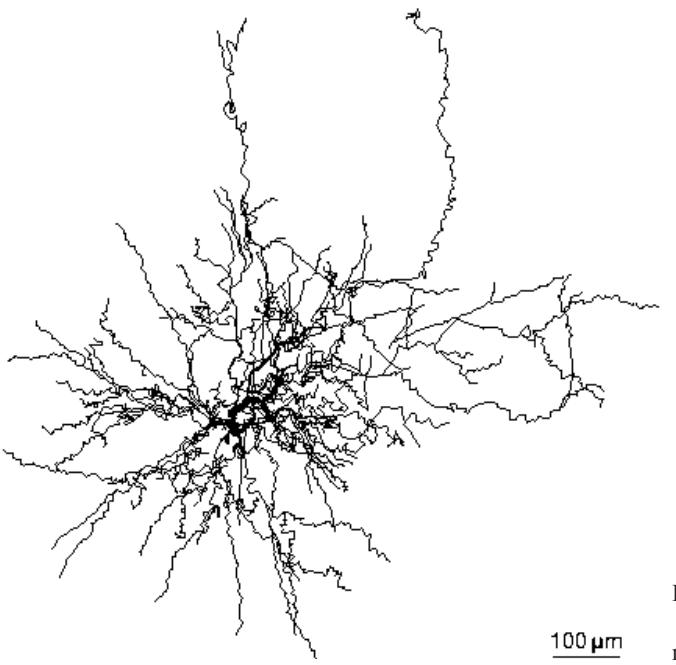
## 2.2. Jak można zrobić sztuczny neuron?

Podstawowym „budulcem", z którego korzystamy przy tworzeniu sieci neuronowej, są sztuczne neurony. Spróbujemy je teraz poznać nieco dokładniej. W poprzednim rozdziale widziałeś już kilka obrazków ilustrujących kształt typowego biologicznego neuronu, ale dla przypomnienia jeszcze jedna ilustracja nie zaszkodzi, więc zobacz na rysunku 2.4, jak wygląda (w uproszczeniu) przykładowy neuron.



Rys. 2.4. Orientacyjna budowa biologicznej komórki nerwowej (neuronu)

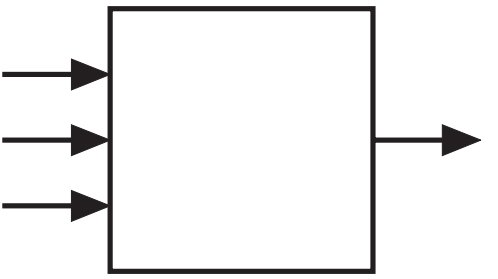
Żebyś nie sądził, że wszystkie prawdziwe neurony wyłącznie tak właśnie wyglądają, na rysunku 2.5 pokazuję Ci jeszcze jeden rysunek prawdziwego biologicznego neuronu, wypreparowanego z kory mózgowej szczura.




Rys. 2.5. Widok preparatu mikroskopowego rzeczywistego neuronu

Trudno na tym rysunku domyślić się, które z licznych widocznych na nim włókien jest **aksonem**, który jest zawsze pojedynczy i jako jedyny wyprowadza sygnały **od** danego neuronu do wszystkich innych, a które pełnią rolę **dendrytów**. Jednak jest to także prawdziwy biologiczny neuron, a więc także i taką komórkę musi dobrze odwzorowywać nasz sztuczny neuron, którym się teraz dokładniej zajmiemy.

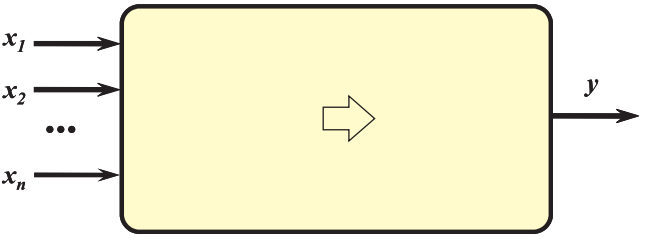
Sztuczne neurony budujące wykorzystywane w technice sieci są oczywiście bardzo uproszczonymi modelami komórek nerwowych występujących w przyrodzie. Budowę sztucznego neuronu najlepiej ilustruje schemat przedstawiony na rysunku 2.6. Porównując ten rysunek z rysunkami 2.4 czy 2.5, uświadomisz sobie poglądowo, jak bardzo twórcy sieci neuronowych upraszczają biologiczną rzeczywistość.



Rys. 2.6. Ogólny schemat sztucznego neuronu pokazuje stopień jego uproszczenia

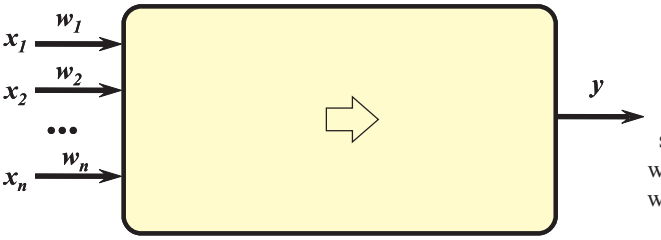
Jednak mimo tych uproszczeń sztuczne neurony zachowują wszystkie te cechy, które są ważne z punktu widzenia zadań, jakie chcemy im powierzyć w budowanych sieciach, będących narzędziami informatyki, a nie modelami biologii:

- Po pierwsze – c**harakteryzują się one występowaniem wielu wejść i jednego wyjścia**. Sygnały wejściowe $x_i$ ($i = 1, 2, \ldots, n$) oraz sygnał wyjściowy $y$ mogą przyjmować wyłącznie wartości liczbowe, najczęściej z zakresu od 0 do 1 (czasem także od –1 do +1), natomiast fakt, że w rozwiązywanych przez sieć zadaniach odpowiadają one pewnym informacjom (na przykład na wyjściu decyzji, jaką osobę rozpoznała sieć neuronowa analizująca czyjąś fotografię), jest wynikiem specjalnej **umowy**. Najczęściej konkretne znaczenia przypisuje się do sygnałów wejściowych i wyjściowych sieci w taki sposób, że najistotniejsze jest to, na którym wejściu lub wyjściu określony sygnał się pojawił (każde wejście i każde wyjście związane jest z określonym **znaczeniem** sygnału), zaś dodatkowo stosuje się **skalowanie** sygnałów, tak dobrane, żeby wartości sygnałów, jakie będą w sieci cyrkulowały, nie wykraczały poza ustalony zakres – na przykład od 0 do 1.
- Po drugie – sztuczne neurony wykonują określone czynności na sygnałach, które otrzymują na wejściach, w wyniku czego produkują sygnały (tylko jeden dla każdego pojedynczego neuronu), które są obecne na ich wyjściach i są przesyłane dalej (do innych neuronów albo na wyjście sieci, jako rozwiązanie postawionego problemu). Zadanie sieci, sprowadzone do funkcjonowania jej podstawowego elementu, jakim jest neuron, polega więc na tym, że przetwarza on informacje wejściowe $x_i$ na wynik $y$ stosując reguły wynikające z tego, jak jest zbudowany, oraz z tego, czego został nauczony. Omówione do tej chwili właściwości neuronu przedstawia rysunek 2.7.




Rys. 2.7. Podstawowe sygnały występujące w neuronie

- Po trzecie – neurony mogą się uczyć. Służą do tego współczynniki $w_i$ nazywane **wagami synaptycznymi**. Jak zapewne pamiętasz z poprzedniego rozdziału – są one odzwierciedleniem dosyć złożonych procesów biochemicznych i bioelektrycznych zachodzących w rzeczywistych synapsach neuronów biologicznych. Z punktu widzenia dalszych rozważań najważniejsze jest to, że wagi synaptyczne mogą podlegać modyfikacjom (czyli mieć zmieniane

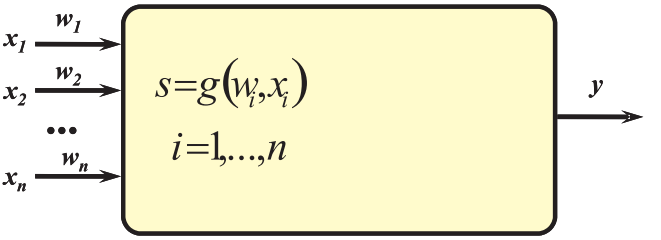



Rys. 2.8. Dodanie do struktury neuronu nastawialnych współczynników wag czyni z niego jednostkę uczącą się

wartości), co stanowi podstawę uczenia sieci. Schemat neuronu zdolnego do uczenia przedstawiono na rysunku 2.8.

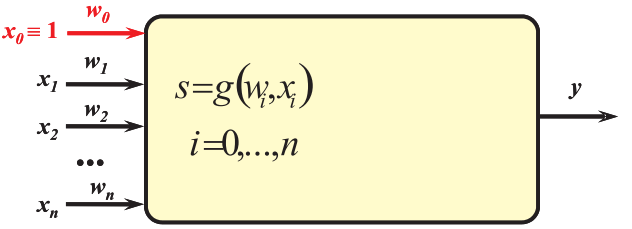
Podsumowując te rozważania można stwierdzić, że sztuczne neurony traktować można jako elementarne **procesory** o następujących właściwościach:

- każdy neuron otrzymuje **wiele sygnałów wejściowych** $x_i$ i wyznacza na ich podstawie swoją „odpowiedź" $y$, to znaczy produkuje **jeden sygnał wyjściowy;**
- z każdym oddzielnym wejściem neuronu związany jest parametr nazywany **wagą** (w literaturze angielskiej *weight*) $w_i$. Nazwa ta oznacza, że wyraża on stopień ważności informacji docierających do neuronu tym właśnie wejściem;
- **sygnał** wchodzący określonym wejściem jest najpierw **modyfikowany** z wykorzystaniem wagi danego wejścia. Najczęściej modyfikacja polega na tym, że sygnał jest po prostu **przemnażany** przez wagę danego wejścia, w związku z czym w dalszych obliczeniach uczestniczy już w formie zmodyfikowanej: **wzmocnionej** (gdy waga jest większa od 1) lub **stłumionej** (gdy waga ma wartość mniejszą od 1). Sygnał z danego wejścia może wystąpić nawet w formie **przeciwstawnej** w stosunku do sygnałów z innych wejść, gdy jego waga ma wartość ujemną. Wejścia z ujemnymi wagami są wśród użytkowników sieci neuronowych określane jako tzw. **wejścia hamujące**, natomiast te z wagami dodatnimi są w związku z tym nazywane **wejściami pobudzającymi**;
- sygnały wejściowe (zmodyfikowane przez odpowiednie wagi) są w neuronie **agregowane** (patrz rys. 2.9). Znowu rozważając sieci w ogólny sposób można by było podawać tu różne sposoby agregacji sygnałów wejściowych, jednak najczęściej polega ona na tym, że odpowiednie sygnały są po prostu **sumowane**, dając w wyniku pewien pomocniczy sygnał wewnętrzny, nazywany łącznym pobudzeniem neuronu albo pobudzeniem postsynaptycznym. W literaturze angielskiej dla określenia tego sygnału bywa używany termin *net value*;
- do tak utworzonej sumy sygnałów neuron dodaje niekiedy (nie we wszystkich typach sieci, ale generalnie często) pewien dodatkowy składnik



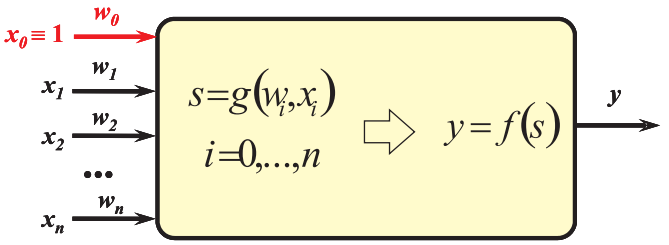

Rys. 2.9. Agregacja danych wejściowych jako pierwsza z wewnętrznych funkcji neuronu

niezależny od sygnałów wejściowych, nazywany **progiem** (w literaturze angielskiej nazywany *bias*). Próg, jeśli jest uwzględniany, także podlega procesowi uczenia, dlatego czasem można sobie wyobrazić, że próg jest dodatkową wagą synaptyczną związaną z wejściem, na które podłączony jest wewnętrzny sygnał o wartości stale wynoszącej 1. Rola progu polega na tym, że dzięki jego obecności właściwości neuronu mogą być kształtowane w trakcie procesu uczenia w sposób znacznie bardziej swobodny (bez jego uwzględnienia charakterystyka funkcji agregacji zawsze musi przechodzić przez początek układu współrzędnych, co stanowi niekiedy uciążliwą „kotwicę"). Schemat neuronu, w którym uwzględniono próg, pokazano na rysunku 2.10;



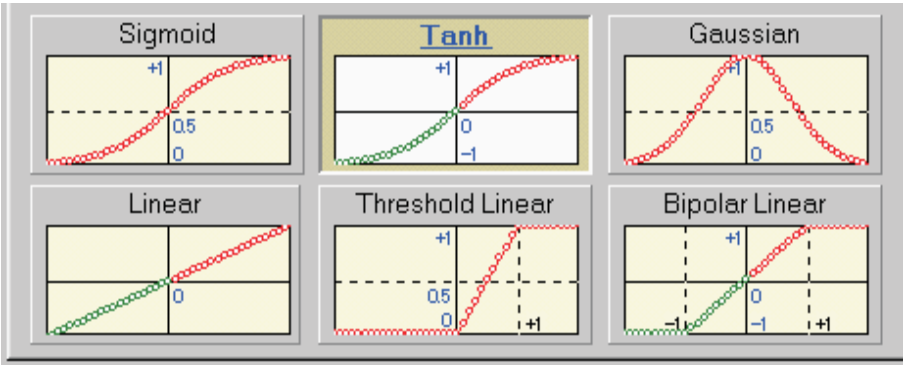

Rys. 2.10. Zastosowanie dodatkowego parametru, jakim jest próg (*bias*)

- suma przemnożonych przez wagi sygnałów wewnętrznych z dodanym (ewentualnie) progiem może być niekiedy bezpośrednio przesyłana do jego aksonu i traktowana jako **sygnał wyjściowy** neuronu. W wielu typach sieci to wystarczy. W taki sposób działają tak zwane sieci liniowe (na przykład sieci nazywane ***ADALINE*** = *ADaptive LINEar*). Natomiast w sieciach o bogatszych możliwościach (na przykład w bardzo popularnych sieciach nazywanych ***MLP*** od słów *Multi-Layer Perceptron*) sygnał wyjściowy neuronu obliczany jest za pomocą pewnej nieliniowej funkcji. Funkcję tę w całej książce oznaczać będziemy symbolem *f( )* albo φ( ). Schemat neuronu zawierającego zarówno agregację sygnałów wejściowych, jak i generację sygnału wyjściowego przedstawia rysunek 2.11;
- funkcja φ( ) zwana jest **charakterystyką neuronu** (w angielskiej literaturze funkcjonuje termin *transfer function*). Znane są różne charakterystyki neuronu, co ilustruje rysunek 2.12. Niektóre z nich są tak wybrane, żeby




Rys. 2.11. Komplet wewnętrznych funkcji neuronu

zachowanie neuronu sztucznego maksymalnie przypominało zachowanie rzeczywistego neuronu biologicznego (funkcja sigmoidalna), ale mogą być one dobrane w taki sposób, żeby zapewniać maksymalną sprawność obliczeń przeprowadzanych przez sieć neuronową (funkcja Gaussa). We wszystkich przypadkach funkcja φ( ) stanowi ważny element pośredniczący między łącznym pobudzeniem neuronu a jego sygnałem wyjściowym;

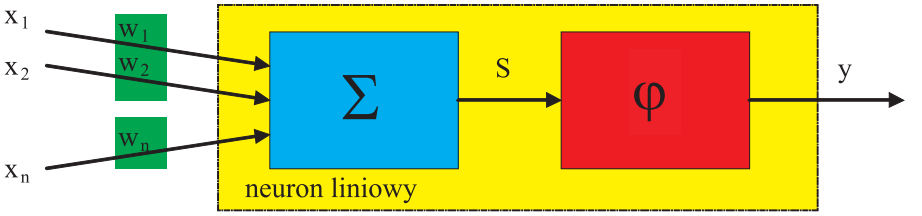



Rys. 2.12. Niektóre częściej używane charakterystyki neuronu

• znajomość sygnałów wejściowych, współczynników wag, sposobu agregacji wejść oraz charakterystyki neuronu, pozwala w każdej chwili jednoznacznie określić jego sygnał wyjściowy, przy czym zwykle zakłada się, że (w odróżnieniu od tego, co dzieje się w rzeczywistych neuronach) proces ten zachodzi **bezzwłocznie**. Dzięki temu w sztucznych sieciach neuronowych zmiany sygnałów wejściowych praktycznie natychmiast uwidaczniane są na wyjściu. Oczywiście, jest to założenie czysto teoretyczne, gdyż nawet przy realizacji elektronicznej konieczny jest pewien czas do tego, żeby po zmianie sygnałów wejściowych odpowiedni układ scalony ustalił właściwą wartość sygnału wyjściowego. Znacznie więcej czasu potrzeba na to, żeby ten sam efekt zrealizować w sieci działającej jako model symulacyjny, bo komputer naśladujący działanie sieci musi wtedy przeliczyć wszystkie wartości wszyst-

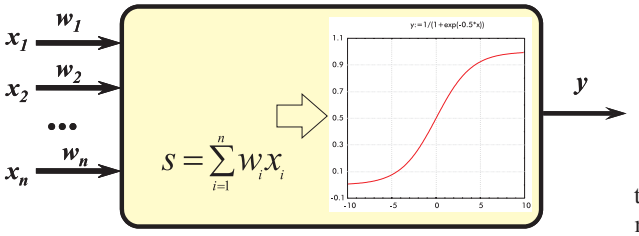
kich sygnałów na wyjściach wszystkich neuronów sieci, co nawet przy bardzo szybkich komputerach może trwać dosyć długo. Mówiąc o bezzwłocznym działaniu neuronów mam na myśli to, że rozważając działanie sieci nie będziemy zwracać uwagi na czynnik, jakim jest czas reakcji neuronu, bo będzie on dla nas nieistotny.

Kompletna opisana struktura pojedynczego neuronu przedstawiona jest na rysunku 2.13.



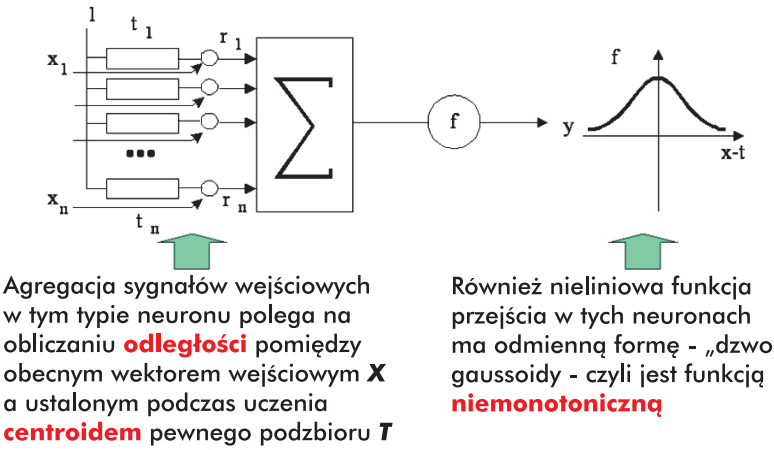

Rys. 2.13. Struktura neuronu jako procesora będącego podstawą budowy sieci neuronowych

Neuron przedstawiony na tym rysunku jest najbardziej typowym „budulcem", jakiego używa się przy tworzeniu sieci neuronowych. Dokładniej – takim typowym „budulcem" jest neuron sieci określanej jako MLP (*Muliti-Layer Perceptron*), którego najważniejsze elementy zebrałem razem i pokazałem na rysunku 2.14. Widać na tym rysunku, że neuron MLP cechuje się funkcją agregacji polegającą na prostym sumowaniu przemnożonych przez „wagi" sygnałów wejściowych oraz korzysta z nieliniowej funkcji przejścia o charakterystycznym kształcie sigmoidy.




Rys. 2.14. Najczęściej spotykany składnik sieci neuronowych – neuron typu MLP

Jednak czasami, w specjalnych celach używane są w sieciach neuronowych tak zwane **neurony radialne**. Mają one nietypową metodę agregacji danych wejściowych, używają też nietypowej charakterystyki (Gaussowskiej) oraz są w nietypowy sposób uczone. Nie chcę w tym momencie rozwijać tematu tych specyficznych neuronów, wykorzystywanych głównie do tworzenia specjalnych sieci określanych jako RBF (od angielskiego *Radial Basis Functions*), ale na rysunku 2.15 przedstawię schemat takiego neuronu radialnego,

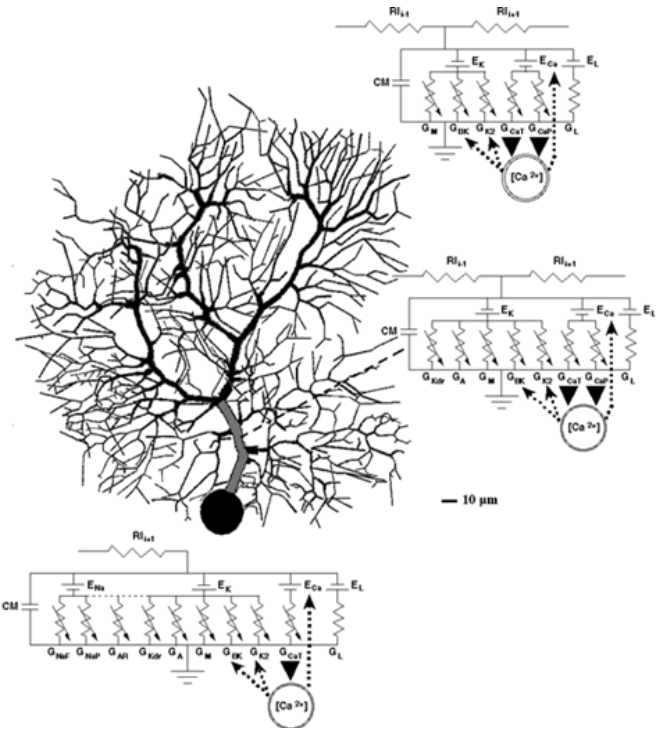



Rys. 2.15. Struktura i osobliwe właściwości neuronu radialnego, oznaczanego też RBF

żeby umożliwić Ci porównanie z wcześniej omówionym typowym neuronem przedstawionym na rysunku 2.14.

## 2.3. Dlaczego nie używamy dokładnego modelu biologicznego neuronu?

Wszystkie sztuczne neurony, sigmoidalne i radialne, opisane w tym rozdziale, a także używane w dalszych częściach tej książki, są **uproszczonymi** modelami rzeczywistych biologicznych neuronów. Stwierdzenie to pojawiało się już kilkakrotnie, teraz jednak chcę Ci pokazać, **jak bardzo uproszczone są sztuczne neurony.** Posłużę się w tym celu przykładem badań prowadzonych przez de Schuttera. Otóż badacz ten przez wiele lat zajmował się tym, żeby maksymalnie wiernie i maksymalnie dokładnie odtworzyć w modelu komputerowym wszystko to, co wiemy na temat struktury i działania (w najdrobniejszych szczegółach!) **jednego** tylko neuronu – konkretnie tak zwanej komórki Purkinjego. Jego model odwoływał się do układów elektrycznych, które zgodnie z badaniami Hodgkina i Huxleya (nagroda Nobla z 1963 roku) modelują bioelektryczną aktywność poszczególnych włókien (dendrytów i aksonu) oraz błony komórkowej ciała neuronu. W badaniach odtworzono z niezwykłą dokładnością kształt rzeczywistej komórki Purkinjego oraz uwzględniono wyniki badań Nehera i Sakmanna (nagroda Nobla z 1991 roku) na temat funkcjonowania tak zwanych kanałów jonowych. Struktura modelowanej komórki oraz schematy elektrycznych układów zastępczych, wykorzystywane w modelu de Schuttera pokazane są na rysunku 2.16.




Rys. 2.16. Model neuronu maksymalnie wierny biologicznemu oryginałowi, używany w badaniach de Schuttera

Model, który zbudował de Schutter, okazał się niesłychanie skomplikowany i kosztowny obliczeniowo. Wystarczy powiedzieć, że do zbudowania modelu użyto:

⇒ 1600 tzw. ***kompartmentów*** (fragmentów komórki traktowanych jako jednorodne struktury zawierające określone związki chemiczne w określonych stężeniach),

⇒ 8021 modeli kanałów jonowych,

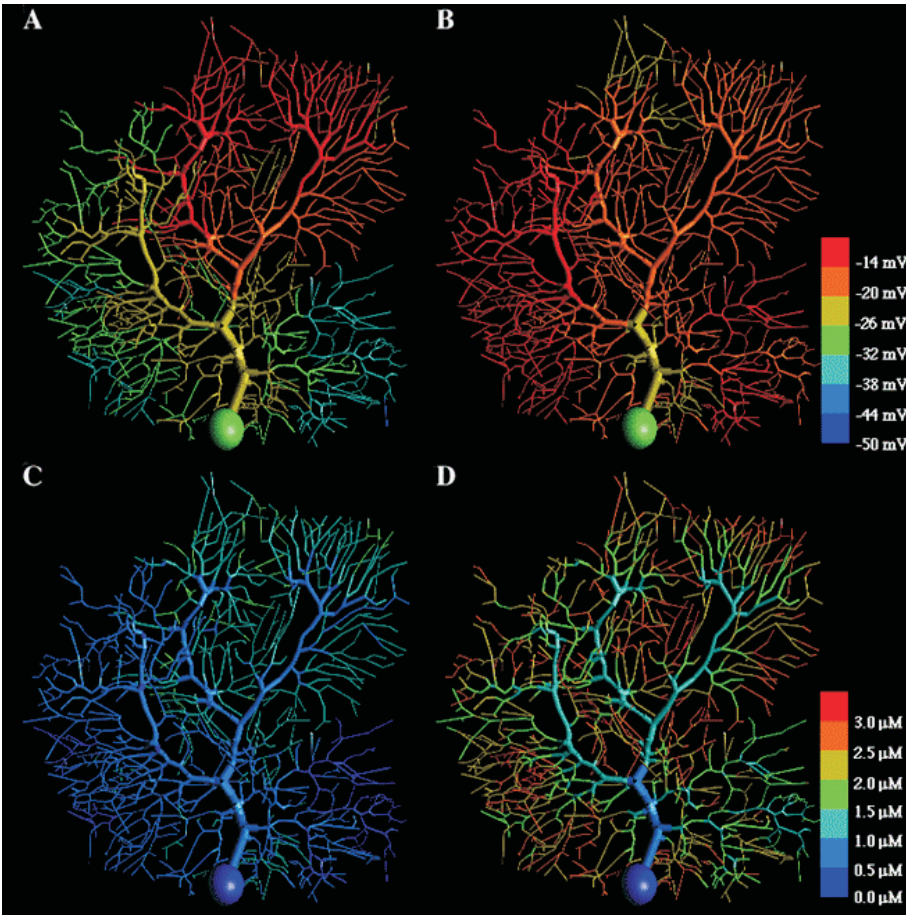
⇒ 10 typów różnych złożonych opisów matematycznych kanałów jonowych zależnych od napięcia,

⇒ **32000 równań różniczkowych (!)**,

⇒ **19200 parametrów** koniecznych do oszacowania przy dostrajaniu modelu,

⇒ dokładnego opisu morfologii komórki zrekonstruowanej za pomocą mikroskopu.

Nic dziwnego, że do zasymulowania kilkunastu sekund „życia" takiej komórki nerwowej konieczne było użycie wielkiego superkomputera, a i tak wy-




Rys. 2.17. Wybrane wyniki uzyskane w badaniach de Schuttera. U góry aktywność elektryczna symulowanej komórki, u dołu zjawiska biochemiczne (przepływ jonów wapnia)

magało to wielu godzin jego nieprzerwanej pracy. Trzeba przyznać, że wyniki tego modelowania są bardzo efektowne. Niektóre z nich przedstawione są na rysunku 2.17.

Jednak wnioski z tych doświadczeń są jednoznaczne: Próba wiernego modelowania struktury i działania rzeczywistego biologicznego neuronu okazała się udana, ale jest to droga zbyt kosztowna, żeby w ten sposób próbować tworzyć **użyteczne praktycznie** sieci neuronowe. Dlatego będziemy od tej chwili już stale używali wyłącznie modeli uproszczonych, oczekując, że mimo tych zastosowanych uproszczeń sieć neuronowa będzie mogła nie tylko skutecznie rozwiązywać różne stawiane jej zadania, ale dodatkowo będzie także zdolna

do tego, że jej zachowania mogą nam nasuwać ciekawe wnioski na temat zachowania ludzkiego (na przykład Twojego!) mózgu. Wkrótce się o tym sam przekonasz!

## 2.4. Jak działa sieć zbudowana ze sztucznych neuronów?

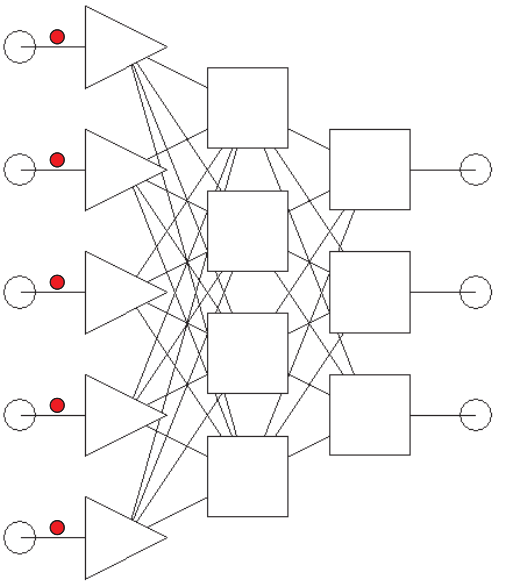
Z przytoczonego wyżej opisu wynika, że każdy neuron dysponuje pewną **wewnętrzną pamięcią** (reprezentowaną przez aktualne wartości wag i progu) oraz pewnymi możliwościami **przetwarzania** wejściowych sygnałów w sygnał wyjściowy. Chociaż możliwości te są dość ograniczone (dzięki temu właśnie neuron jest stosunkowo mało kosztownym procesorem i można zbudować system zawierający setki czy tysiące takich elementów), to jednak okazują się one wystarczające do zbudowania systemów realizujących bardzo złożone zadania przetwarzania danych.

Na skutek tego, że zarówno zasób informacji gromadzonych przez każdy pojedynczy neuron sieci jest ograniczony (no bo jeden neuron ma na ogół niewiele nastawialnych wag), jak i z powodu bardzo ubogich możliwości obliczeniowych takiego pojedynczego neuronu (tylko agregacja sygnałów w wyliczenie sygnału wyjściowego) – **sieć neuronowa z reguły musi zawierać wiele neuronów i może działać wyłącznie jako całość.** Jak z tego wynika, wszystkie omówione w poprzednim rozdziale możliwości i właściwości sieci neuronowych są wynikiem **kolektywnego działania** bardzo wielu połączonych ze sobą elementów (całej sieci, a nie pojedynczych neuronów), w związku z czym uzasadniona jest spotykana niekiedy nazwa ***MPP*** *(Massive Parallel Processing),* jaką wiąże się z całą tą dziedziną informatyki.

Zainteresujmy się na chwilę funkcjonowaniem całej sieci neuronowej. Jak wynika z przytoczonych wyżej uwag, zarówno **program** działania oraz informacje stanowiące **bazę wiedzy**, a także **dane**, na których wykonuje się obliczenia, jak i sam **proces obliczania** – są w sieci całkowicie **rozproszone**. Nie da się wskazać miejsca, w którym zgromadzona jest w niej taka lub inna konkretna informacja, chociaż sieci bywają wykorzystywane jako pamięci, zwłaszcza tzw. pamięci skojarzeniowe i bardzo dobrze spełniają swoje zadania. Podobnie niemożliwe jest zlokalizowanie w określonym miejscu sieci jakiegoś wydzielonego fragmentu wykonywanego przez nią algorytmu, na przykład wskazanie, które elementy sieci odpowiedzialne są za wstępne przetwarzanie i analizę wprowadzanych danych, a które dostarczają końcowego rozwiązania podawanego przez sieć.

Popatrzmy, jak to działa i jaka jest rola poszczególnych składników w funkcjonowaniu całej sieci. W rozważaniach tu prowadzonych założymy, że sieć **ma** już ustalone wszystkie współczynniki wagowe, czyli że proces

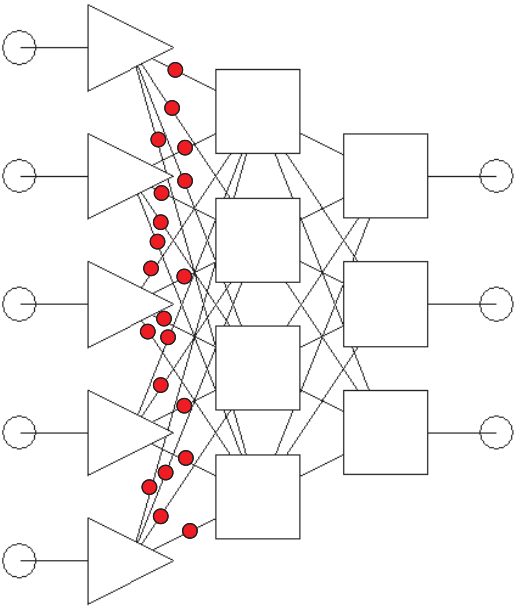
jej uczenia został zakończony. Uczeniem sieci, które jest bardzo ważne, ale i trudniejsze do prześledzenia, zajmiemy się bowiem osobno w następnych rozdziałach. Zaczniemy naszą analizę od momentu, kiedy sieci postawione zostaje nowe zadanie do rozwiązania. Zadanie to niosą sygnały wejściowe, które pojawiają się na wszystkich wejściach sieci. Na rysunku 2.18 sygnały te symbolizują czerwone punkty ulokowane w odpowiednich miejscach schematu sieci.



Rys. 2.18. Początek działania sieci neuronowej wiąże się z pojawieniem się na jej wejściach sygnałów (czerwone kropki) niosących nowe zadanie do rozwiązania

Sygnały wejściowe trafiają do neuronów warstwy wejściowej, które ich z reguły **nie przetwarzają**, tylko dokonują ich dystrybucji – to znaczy rozsyłają do wszystkich neuronów warstwy ukrytej (rys. 2.19). Na marginesie warto zauważyć, że odmienna rola neuronów warstwy wejściowej, które nie przetwarzają sygnałów, tylko je rozsyłają, akcentowana bywa na rysunkach przedstawiających sieci neuronowe odmiennym kształtem symboli graficznych oznaczających te neurony na schematach (na przykład używany jest trójkąt zamiast kwadratu, jak na przedstawianych tu rysunkach).

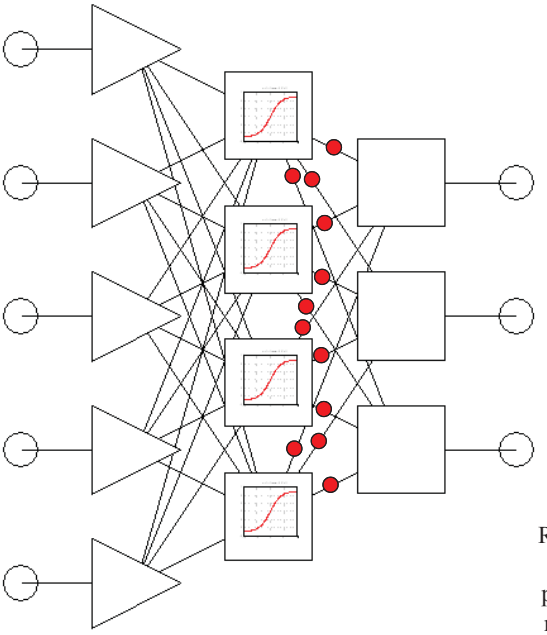
Kolejny etap polega na aktywizacji neuronów warstwy ukrytej. Neurony te wykorzystując swoje wagi (czyli angażując zgromadzoną w nich wiedzę) najpierw modyfikują sygnały wejściowe, następnie agregują je, a potem korzystając ze swoich charakterystyk (na rysunku 2.20 przedstawionych jako funkcje sigmoidalne) wyliczają sygnały wyjściowe, które zostają skierowane do neuronów warstwy wyjściowej. Ten etap przetwarzania informacji w sieci



Rys. 2.19. Sygnały wejściowe (nie przetworzone w żaden sposób w warstwie wejściowej) są rozsyłane do wszystkich neuronów warstwy ukrytej

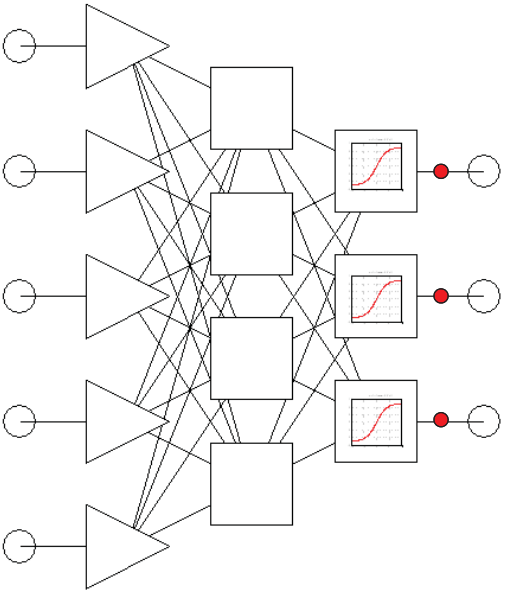
neuronowej ma szczególnie duże znaczenie. Warstwy ukrytej wprawdzie nie widać z zewnątrz sieci (jej sygnałów nie da się zaobserwować ani na wejściu, ani na wyjściu sieci, stąd taka właśnie, a nie inna jej nazwa), ale to tutaj wykonywana jest większa część pracy, związanej z rozwiązywaniem zadania. Najwięcej połączeń i najwięcej związanych z nimi współczynników wagowych znajduje się zwykle właśnie pomiędzy warstwą wejściową a warstwą ukrytą, stąd można powiedzieć, że większość wiedzy, zgromadzonej w trakcie uczenia, lokuje się właśnie w tej warstwie. Sygnały produkowane przez neurony warstwy ukrytej nie mają żadnej bezpośredniej interpretacji, w odróżnieniu od sygnałów wejściowych i wyjściowych, z których każdy coś konkretnego znaczy w kontekście rozwiązywanego zadania. Jednak posługując się analogią z procesem produkcyjnym można powiedzieć, że neurony warstwy ukrytej wytwarzają „półprodukty", czyli sygnały w taki sposób charakteryzujące rozwiązywane zadanie, że potem stosunkowo łatwo jest z ich pomocą „zmontować końcowy produkt", czyli znaleźć ostateczne rozwiązanie w neuronach warstwy wyjściowej (rys. 2.20).

Śledząc dokładniej działanie sieci na tym końcowym etapie rozwiązywania postawionego problemu możemy zauważyć, że neurony warstwy wyjściowej wykorzystują swoje możliwości agregacji sygnałów oraz swoje charakterystyki żeby zbudować końcowe rozwiązanie, podawane na wyjściu sieci (rys. 2.21).



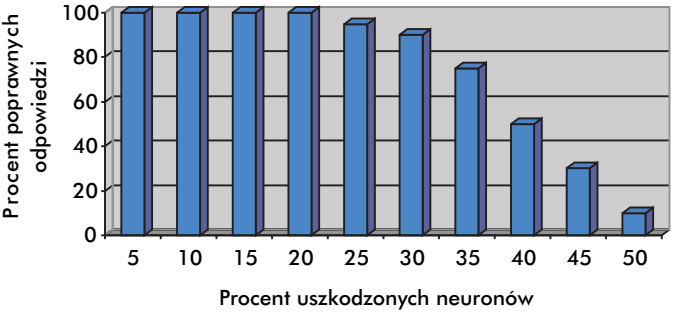
Rys. 2.20. Po przetworzeniu sygnałów przez neurony warstwy ukrytej powstają sygnały pośrednie, kierowane do neuronów warstwy wyjściowej

Powtórzymy jeszcze raz: **sieć działa zawsze jako całość** i wszystkie jej elementy mają swój wkład w realizację wszystkich czynności, które sieć realizuje – podobnie jak się to dzieje przy odtwarzaniu hologramu, gdzie z każ-



Rys. 2.21. Neurony warstwy wyjściowej korzystają ze wstępnie opracowanej informacji pochodzącej z warstwy ukrytej i obliczają końcowe wyniki, będące rozwiązaniem postawionego zadania

dego kawałka rozbitej płyty fotograficznej można odtworzyć cały obraz sfotografowanego (sholografowanego?) przedmiotu. Jedną z konsekwencji takiego działania sieci jest jej niewiarygodna zdolność do poprawnego działania nawet po uszkodzeniu znacznej części wchodzących w jej skład elementów. Był taki badacz sieci neuronowych (*Frank Rosenblatt*), który najpierw uczył budowane przez siebie sieci neuronowe jakiejś umiejętności (na przykład rozpoznawania obrazów liter), a potem egzaminował uszkadzając przy tym celowo coraz większą liczbę ich elementów (sieci były realizowane jako specjalistyczne układy elektroniczne). Okazywało się, że można było uszkodzić sporą część elementów sieci, a ona mimo to prawidłowo wykonywała swoje czynności (rys. 2.22). Uszkodzenie większej liczby neuronów i połączeń powodowało wprawdzie pogorszenie jakości działania sieci, ale polegało ono na tym, że




Rys. 2.22. Sieć neuronowa ma zdumiewającą własność: może działać poprawnie nawet wtedy, gdy uszkodzone zostanie sporo jej elementów składowych!

uszkodzona sieć coraz częściej myliła się przy rozpoznawaniu (wskazywała na przykład D, gdy pokazywano jej literę O) – ale nie odmawiała pracy. Porównaj to zachowanie sieci ze znanym Ci faktem, że w większości innych urządzeń elektronicznych (komputer, telewizor) wystarczy uszkodzenie jednego ważnego elementu, by system przestał działać jako całość, a następnie weź pod uwagę znany fakt, że w mózgu dorosłego człowieka **codziennie** ginie (z różnych względów) kilka tysięcy neuronów – a jednak nasz mózg jako całość działa niezawodnie przez wiele lat.

## 2.5. Jak struktura sieci neuronowej wpływa na to, co sieć może zrobić?

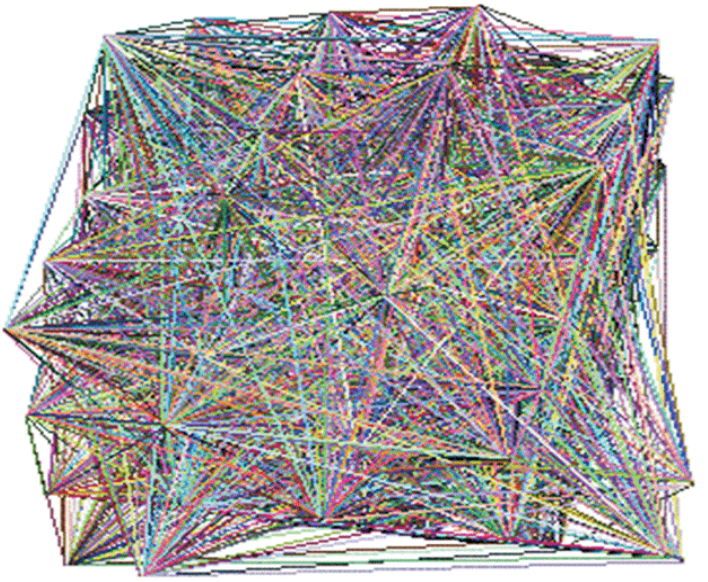
Rozważmy teraz związek, jaki zachodzi pomiędzy strukturą sieci a zadaniami, które może ona wykonywać. Jak już wiesz, z neuronów o własnościach omówionych w poprzednim podrozdziale tworzona jest sieć. Struktu-

ra tej sieci powstaje w ten sposób, że wyjścia jednych neuronów łączy się (według wybranego schematu) z wejściami innych, tworząc łącznie system zdolny do równoległego, w pełni współbieżnego przetwarzania różnych informacji. Z powodów, o których także była już mowa, wybieramy zwykle sieć o strukturze warstwowej, a połączenia pomiędzy neuronami poszczególnych warstw kształtujemy przyjmując zasadę połączeń typu „każdy z każdym". Oczywiście, konkretna topologia sieci, to znaczy głównie liczba wybranych neuronów w poszczególnych warstwach, powinna wynikać z rodzaju zadania, jakie zamierzamy sieci postawić. W teorii reguła jest dosyć prosta: Im bardziej skomplikowane zadanie, tym więcej neuronów musi mieć sieć, żeby go rozwiązać, bo sieć mająca większą liczbę neuronów jest po prostu inteligentniejsza. W praktyce jednak nie jest to wcale tak jednoznaczne, jak by się mogło wydawać.

W bogatej literaturze dotyczącej sieci znaleźć bowiem można liczne prace, w których wykazano, że w istocie **decyzje dotyczące struktury sieci wpływają na jej zachowanie znacznie słabiej, niż by można było oczekiwać**. To paradoksalne stwierdzenie wynika z faktu, że zachowanie sieci w zasadniczy sposób determinowane jest przez proces jej **uczenia**, a nie przez **strukturę** czy liczbę użytych do jej budowy elementów. Oznacza to, że sieć mająca zdecydowanie gorszą strukturę może znacznie skuteczniej rozwiązywać postawione zadania (gdy zostanie dobrze nauczona) niż sieć o optymalnie dobranej strukturze, ale źle trenowana. Znane są doświadczenia, w których strukturę sieci wybierano całkowicie **przypadkowo** (ustalając na drodze losowania, które elementy należy ze sobą połączyć i w jaki sposób), a sieć mimo to zdolna była do rozwiązywania stawianych jej trudnych zadań!

Przyjrzyjmy się uważniej konsekwencjom tego ostatniego stwierdzenia, są one bowiem dosyć ważne i ciekawe. Skoro sieć mogła osiągnąć prawidłowe wyniki, chociaż jej strukturę zaprojektowano całkowicie w sposób przypadkowy, to oznacza to, że proces uczenia mógł tak każdorazowo dostosować jej parametry do wymaganych operacji, wynikających z realizacji ustalonego algorytmu, że proces rozwiązywania zadania przebiegał poprawnie mimo całkowicie **losowej** struktury sieci. Te doświadczenia, wykonane po raz pierwszy przez wspomnianego już wyżej *Franka Rosenblatta* na początku lat 70., były bardzo efektowne: badacz rzucał kostką albo wyciągał losy – i w zależności od tego, co mu z tych losowań wychodziło, łączył pewne elementy sieci ze sobą albo nie. Powstająca struktura połączeń była całkowicie **chaotyczna** (patrz rysunek 2.23) – a jednak sieć po procesie uczenia potrafiła bardzo sensownie wykonywać stawiane jej zadania.

Wyniki tych badań, relacjonowane przez Rosenblatta w jego publikacjach, były tak zadziwiające, że początkowo uczeni nie wierzyli, że coś takiego jest



Rys. 2.23. Struktura połączeń sieci, której elementy były ze sobą łączone z wykorzystaniem reguł losowych. Zadziwiające jest to, że taka sieć może (po procesie uczenia) wykazywać skuteczne i celowe działanie

w ogóle możliwe. Jednak potem doświadczenia te były wielokrotnie powtarzane (między innymi w ZSRR akademik Głuszkow specjalnie w tym celu zbudował sieć neuronową, którą nazwał *Alfa*) i dowiodły, że sieć o przypadkowych połączeniach może nauczyć się poprawnego rozwiązywania zadań, chociaż oczywiście proces uczenia takiej sieci jest dłuższy i trudniejszy niż takiej sieci, której struktura sensownie nawiązuje do zadania, które trzeba rozwiązać.

Co ciekawe, wynikami ogłoszonymi przez Rosenblatta zainteresowali się także filozofowie, którzy uznali, że w ten oto sposób została udowodniona pewna teza, pochodząca jeszcze od *Arystotelesa* i rozwinięta potem przez *Locke'a*. Chodzi o koncepcję znaną w filozofii pod nazwą *tabula rasa* – umysłu rodzącego się jako pusta, nie zapisana karta, zapełniana dopiero w trakcie nauki i gromadzenia doświadczeń. Rosenblatt dowiódł, że ta koncepcja jest technicznie możliwa – przynajmniej w formie sieci neuronowej. Odrębnym zagadnieniem jest próba odpowiedzi na pytanie, czy tak właśnie jest z realnym umysłem konkretnego człowieka? Czy istotnie, jak utrzymywał *Locke*, wrodzone uzdolnienia są niczym, a zdobyta w procesie nauki wiedza – wszystkim?

Nie wiemy tego na pewno, ale chyba jednak tak nie jest. Natomiast wiemy z całą pewnością, że sieci neuronowe **mogą** całą swoją wiedzę zyskiwać

wyłącznie w trakcie nauki i nie muszą mieć z góry zadanej, dopasowanej do stawianych im zadań, jakiejkolwiek precyzyjnie określonej struktury. Oczywiście, sieć musi mieć wystarczający stopień złożoności, żeby w jej strukturze można było w toku uczenia „wykrystalizować" potrzebne połączenia i struktury. Zbyt mała sieć nie jest w stanie nauczyć się niczego, gdyż jej „potencjał intelektualny" na to nie pozwala – rzecz jednak nie w strukturze, a w liczbie elementów. Szczura nikt nie uczy teorii względności, chociaż można go wytresować w rozpoznawaniu drogi wewnątrz skomplikowanego labiryntu. Podobnie nikt się nie rodzi „zaprogramowany" do tego, by być wyłącznie genialnym chirurgiem lub koniecznie tylko budowniczym mostów (o tym decydują specjalistyczne studia) – chociaż niektórym ludziom wystarcza intelektu zaledwie do tego, by ładować piasek na ciężarówkę, a i to pod nadzorem. Tak już jest i żadne górnolotne frazesy na temat równości nie są tego w stanie zmienić. Jedni mają wystarczające zasoby intelektualne, inni nie – podobnie jak jedni mają smukłą sylwetkę i okazałą muskulaturę, a inni wyglądają... jakby długo siedzieli przed monitorem komputera.

W przypadku sieci sytuacja jest podobna – nie można spowodować, by sieć z góry miała jakieś szczególne uzdolnienia, można jednak łatwo wyprodukować cybernetycznego kretyna, który nigdy się niczego nie nauczy, bo ma za małe możliwości. Struktura sieci może więc być dowolna, byleby była dostatecznie duża. Wkrótce dowiemy się także, że nie powinna być za duża, bo to także szkodzi – ale o tym porozmawiamy dokładniej za chwilę.

## 2.6. Jak mądrze wybierać strukturę sieci?

Niezależnie od przytoczonych uwag, wskazujących na możliwość osiągnięcia sukcesu poprzez nauczenie określonego działania sieci o strukturze niekoniecznie optymalnie dopasowanej do rozwiązywanego zadania – **jakąś** strukturę trzeba sieci nadać. Ponadto łatwo wykazać, że wybranie na początku **rozsądnej** struktury, dobrze dopasowanej do specyfiki rozwiązywanego zadania, może w istotny sposób skracać czas uczenia i polepszać jego końcowe wyniki. Dlatego pewne uwagi na temat wyboru struktury sieci muszę Ci tu przedstawić, chociaż z pewnością nie będą to uniwersalne recepty na wszystkie możliwe bolączki. Czuję się zobowiązany do podania Ci tutaj kilku wskazówek, ponieważ wszyscy wiemy, jak trudne rozterki wiążą się czasem z wyborem **dowolnego,** nie narzuconego z góry rozwiązania. Postawienie konstruktora sieci w sytuacji, kiedy może przyjąć dowolną jej organizację, jest podobne do dylematów początkujących informatyków, z zakłopotaniem wpatrujących się w pojawiający się niekiedy na ekranie komunikat systemu: *Press any key...*

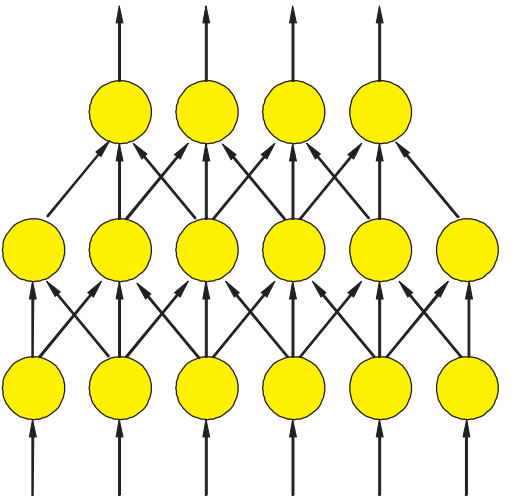
**Który** to jest ten **dowolny** klawisz, który mam nacisnąć?!

Można się z tego śmiać, ale dla mnie podobnie brzmi często spotykane i moich studentów i doktorantów, pełne rozpaczy pytanie: no dobrze, ale **jaka** to jest ta **dowolna** struktura sieci?

Powiem więc teraz kilka słów o możliwych i często spotykanych strukturach sieci, wyraźnie akcentując, że podane niżej informacje i propozycje nie wyczerpują wszystkich możliwości, przeciwnie – każdy badacz może i powinien być tu swoistym Demiurgiem, twórcą i kreatorem nowych bytów, gdyż właściwości sieci o różnych strukturach nie są jeszcze dostatecznie poznane i to jest praca, przy której przyda się każda para... półkul mózgowych.

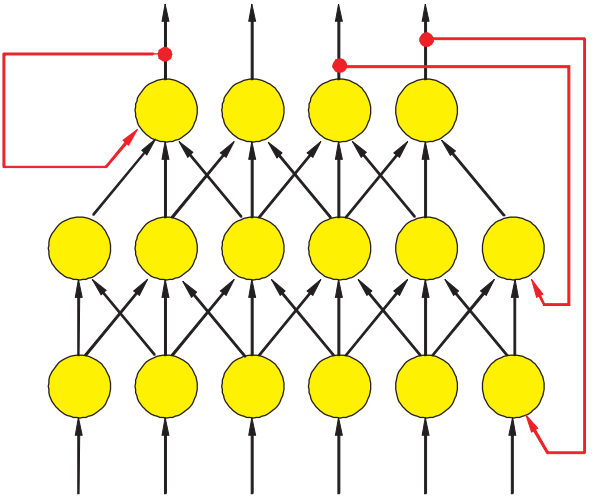
Najpierw dokonam pewnego podziału struktur często stosowanych sieci neuronowych na dwie ważne klasy: z jednej strony rozważać będziemy struktury nie zawierające sprzężeń zwrotnych, a z drugiej struktury, które takie sprzężenia zawierają. Pierwsze z wymienionych sieci określane są często terminem **feedforward**, który jest wprawdzie obco brzmiący i na pozór trochę tajemniczy, ale wygodniejszy od terminu opisowego, jaki by można nadać tym sieciom w naszym języku: „*sieci z jednokierunkowym przepływem sygnału*". Drugie sieci mogą zawierać sprzężenia zwrotne, w których sygnały mogą krążyć dowolnie długo i dlatego bywają nazywane sieciami **rekurencyjnymi**.

- Sieci *feedforward* to struktury, w których istnieje ściśle określony kierunek przepływu sygnałów – od pewnego ustalonego wejścia, na którym podaje się sieci sygnały będące danymi wejściowymi, precyzującymi zadania, które mają być rozwiązywane – do wyjścia, na którym sieć podaje ustalone rozwiązanie (patrz rysunek 2.24). Takie sieci są najczęściej stosowane i najbardziej użyteczne. Ich obszerniejsze omówienie stanowić będzie treść dalszej części tego rozdziału i kilku następnych.



Rys. 2.24. Przykładowa struktura sieci *feedforward*. Neurony, symbolizowane tu przez żółte kółka, łączone są w taki sposób, że przepływ sygnałów możliwy jest wyłącznie od wejścia do wyjścia

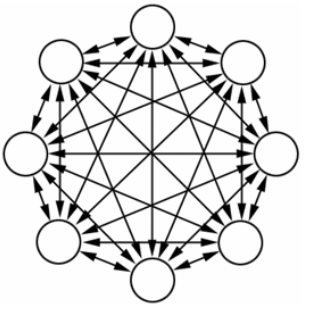
• Sieci *rekurencyjne* cechują się natomiast tym, że neurony tworzą sprzężenia zwrotne, liczne i skomplikowane zamknięte pętle, w których impulsy mogą długo krążyć i zmieniać się, zanim sieć osiągnie pewien stan ustalony – o ile go w ogóle osiągnie (rys. 2.25).



Rys. 2.25. Przykładowa struktura sieci rekurencyjnej. Wyróżniono czerwonym kolorem połączenia będące sprzężeniami zwrotnymi, powodujące, że sieć stała się siecią rekurencyjną

Analiza właściwości i możliwości sieci rekurencyjnych jest znacznie bardziej złożona niż w przypadku sieci feedforward, ale też możliwości obliczeniowe tych sieci są fascynująco odmienne od możliwości innych typów sieci – są one na przykład zdolne do znajdowania rozwiązań problemów typu optymalizacyjnego – szukania najlepszych rozwiązań pewnych klas zadań, czego sieci feedforward robić z reguły nie potrafią.

• Wśród sieci rekurencyjnych szczególne miejsce zajmują sieci związane z nazwiskiem **Hopfielda**, które mają wyjątkowo dużo sprzężeń zwrotnych. Prawdę mówiąc, w tych sieciach **wszystkie** neurony są sprzężone ze sobą na zasadzie sprzężenia zwrotnego i nie ma w tych sieciach żadnych innych połączeń poza sprzężeniami zwrotnymi (rys. 2.26).

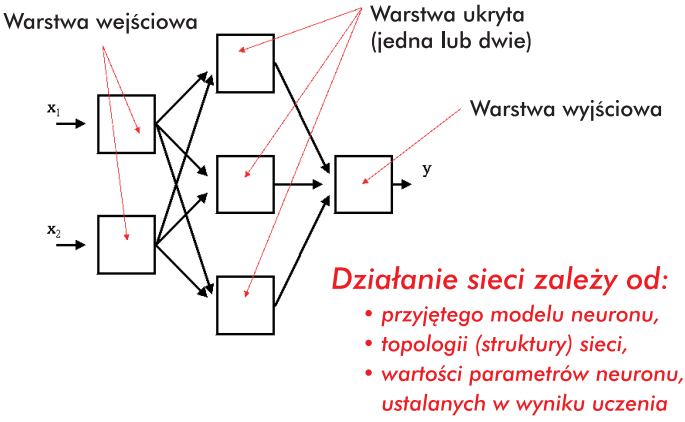


Rys. 2.26. Sieć Hopfielda, w której wszystkie neurony są ze sobą nawzajem powiązane na zasadzie sprzężeń zwrotnych

Swojego czasu prawdziwą sensacją było uzyskanie za pomocą sieci Hopfielda (tej „krańcowo rekurencyjnej") rozwiązania słynnego problemu komiwojażera, co otworzyło dla tych sieci obszerną i ważną klasę problemów obliczeniowych *NP-zupełnych,* o czym jednak opowiem Ci dokładniej przy innej okazji. Jednak mimo tego sensacyjnego osiągnięcia sieci Hopfielda nie stały się tak popularne, jak inne rodzaje sieci, wiec wzmiankujemy o nich dopiero na końcu książki, w rozdziale 11.

Ponieważ budowa sieci ze sprzężeniami zwrotnymi jest bez wątpienia zadaniem trudniejszym i bardziej skomplikowanym niż korzystanie z sieci *feedforward*, a ponadto trudniej jest zapanować nad siecią, w której kłębi się parę tysięcy równoległych, dynamicznych procesów, niż nad siecią, w której sygnały grzecznie i spokojnie przepływają od wejścia na wyjście, dlatego warto **zacząć** znajomość z sieciami neuronowymi właśnie od sieci z jednokierunkowym przepływem sygnałów, przechodząc potem stopniowo i powoli do sieci rekurencyjnych. Jeśli słyszałeś już o najsłynniejszej z nich, sieci Hopfielda i pragniesz z nią zawrzeć znajomość, to albo porzuć czytanie kolejnych rozdziałów i od razu zacznij czytać to, co przygotowałem dla Ciebie w 11. rozdziale, albo (co zdecydowanie polecam) czytaj systematycznie rozdział po rozdziale – ale wtedy musisz się uzbroić w cierpliwość.

Skupiając obecnie uwagę na sieciach feedforward możemy stwierdzić, że do opisania ich struktury stosuje się wygodny i uniwersalny **model warstwowy**. W modelu tym zakłada się, że neurony zgrupowane są w pewne zespoły (warstwy), tak zorganizowane, że główne połączenia i związane z nimi przepływy sygnałów odbywają się pomiędzy elementami sąsiednich warstw. Struktura ta była już omawiana, także i w tym rozdziale, ale nie zaszkodzi przyjrzeć się jej jeszcze raz (rys. 2.27).




Rys. 2.27. Schematyczna budowa najprostszej wielowarstwowej sieci neuronowej

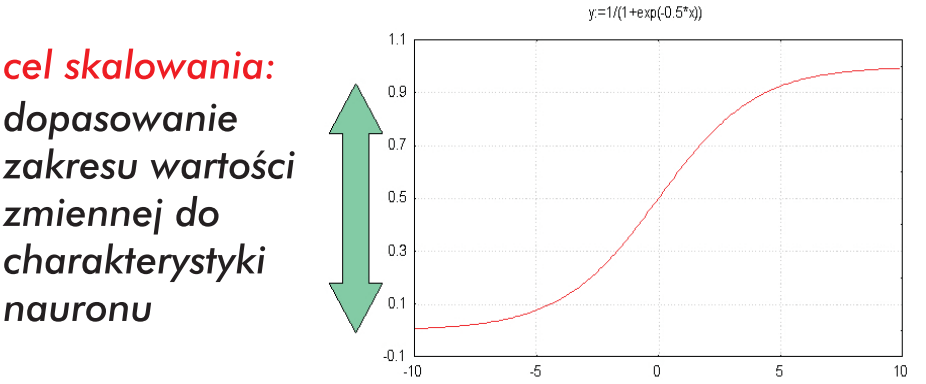
Wspominałem już o tym w poprzednim rozdziale, ale warto może powtórzyć, że połączenia między neuronami sąsiednich warstw mogą być kształtowane na wiele sposobów (wedle uznania twórcy sieci), jednak najczęściej korzysta się ze schematu połączeń typu „każdy z każdym", licząc na to, że proces uczenia doprowadzi do samorzutnego „wykrystalizowania się" potrzebnego zbioru połączeń – po prostu na wejściach, które okażą się zbyteczne z punktu widzenia rozwiązywanego zadania, proces uczenia ustawi współczynniki (wagi) wynoszące zero, co w praktyce przerwie te niepotrzebne połączenia.

## 2.7. Jakimi informacjami karmić sieci?

Wśród warstw neuronów budujących sieć neuronową pierwsza powinna być omówiona **warstwa wejściowa.** Warstwa ta otrzymuje dane z zewnątrz sieci (tą drogą wprowadzane są zadania podlegające rozwiązywaniu). Przy projektowaniu tej warstwy twórca sieci ma tu ułatwioną decyzję – liczba elementów tej warstwy jest bowiem ściśle zdeterminowana przez liczbę danych wejściowych, które trzeba brać pod uwagę przy rozwiązywaniu określonego zadania. Inna rzecz, że czasem samo tylko zdecydowanie się, ile i jakich danych należy wprowadzać do sieci, by poradziła sobie ona ze stawianym zadaniem, nie jest sprawą łatwą. Na przykład, w zadaniu prognozowania kursów akcji na giełdzie – wiadomo, że niektórzy badacze osiągają tu bardzo zachęcające wyniki, przynoszące bardzo duże korzyści inwestorom, którzy w oparciu o produkowane przez sieć prognozy podejmują decyzje o zakupie lub sprzedaży określonych walorów. Jednak publikacje, jakie się na ten temat pojawiają, są bardzo powściągliwe w ujawnianiu, jakie dane stanowiły punkt wyjścia prowadzonych obliczeń. Owszem, mówi się o tym, że stosowano sieć, że ją uczono (podaje się nawet algorytm uczenia), podaje się wyniki (jak wiele zyskano dzięki trafnym inwestycjom, jak dokładnie sieć prognozowała zmiany kursów akcji – tu możliwe są bardzo ładne wykresy linii rzeczywistych zmian i linii prognoz), natomiast w odniesieniu do danych wejściowych mówi się tylko, że podawano informacje dotyczące wcześniejszych zmian notowań akcji oraz wyniki analiz finansowych notowanych na giełdzie spółek. Jak te dane preparowano, które wykorzystano i w jakim stopniu – jakoś autorzy zapominają napisać. Takie roztrzepane gapy!

Z wprowadzeniem danych do sieci neuronowej (a także z uzyskiwaniem od sieci rozwiązań, o czym będzie mowa w kolejnym rozdziale) związana jest jednak pewna subtelność, na którą warto zwrócić uwagę. Otóż neurony dysponują wprawdzie możliwością dostarczania rozwiązania w postaci wartości liczbowej, jednak wartość ta podlega dość istotnym ograniczeniom. Na przykład w większości implementacji sieci sygnały wyjściowe wszystkich

neuronów mogą przyjmować wartości z przedziału od 0 do 1 (lub – co bywa korzystniejsze – od –1 do 1), zatem jeśli potrzebne nam wyniki mają mieć wartości z innego przedziału – konieczne jest pewne **skalowanie** (rys. 2.28).




Rys. 2.28. Zakres wartości rzeczywistej zmiennej, która ma się pojawić na wejściu lub na wyjściu sieci neuronowej, musi być przeskalowany tak, aby mieścił się w przedziale dozwolonym dla neuronu

Problem skalowania danych wejściowych jest w istocie mniej krytyczny niż w przypadku danych wyjściowych (o czym będzie mowa w następnym podrozdziale), bo do wejściowych neuronów można w zasadzie „wepchnąć" na ich wejściach właściwie dowolny sygnał o dowolnej wartości. Na wyjściu jest inaczej, bo neuron nie potrafi wyprodukować innego sygnału niż taki, jaki pozwala mu wytworzyć jego charakterystyka. Jednak dla zachowania jednorodności interpretacji wszystkich cyrkulujących w sieci sygnałów oraz związanych z nimi wag skalowanie wartości wejściowych wykonywane jest prawie zawsze, co ma też dodatkową zaletę, że przy okazji niejako dokonywana jest tzw. **normalizacja** zmiennych wejściowych.

Wyraz „normalizacja" może brzmieć groźnie i sprawiać wrażenie czegoś bardzo skomplikowanego, ale w istocie chodzi o to, żeby zagwarantować sieci „równouprawnienie" jej wszystkich sygnałów wejściowych. Problem polega na tym, że niektóre zmienne wejściowe, niosące informacje istotne dla rozwiązywanego zadania, mogą przyjmować (z samej swojej natury) niewielkie wartości, podczas gdy inne, wcale nie bardziej ważne, mogą przyjmować wartości bardzo duże. Na przykład, w sieci neuronowej, która powinna pomagać lekarzowi w rozpoznaniu choroby, możemy mieć na jednym wejściu podaną ciepłotę ciała pacjenta, a na drugim liczbę erytrocytów (czerwonych krwinek) ustaloną w trakcie analizy jego krwi. Obie te informacje są ważne i o poprawnym rozpoznaniu raz może przesądzić analiza jednej z nich, a innym razem tej

drugiej. Jednak temperatura ciała człowieka wyraża się niewielką liczbą (jak wiesz, u zdrowego człowieka jest to 36,6 stopni Celsjusza) i nawet niewielkie jej zmiany (o kilka stopni w górę lub w dół) mogą znamionować bardzo poważną chorobę. Tymczasem liczba czerwonych krwinek (oznaczana w jednym mililitrze krwi) wynosi około 5 milionów i jej zmiana (nawet o milion) wcale nie jest niczym szczególnie niepokojącym. Gdyby nie przeprowadzono skalowania, to neuron „widzący" na jednym swoim wejściu ogromne wartości liczbowe związane z reprezentacją danych dotyczących analizy krwi – tylko od nich uzależniałby swoje działanie, ignorując całkowicie swoje drugie wejście (to dotyczące temperatury). Dzięki normalizacji obie zmienne stają się w pełni „równouprawnione", a działanie sieci staje się skuteczniejsze.

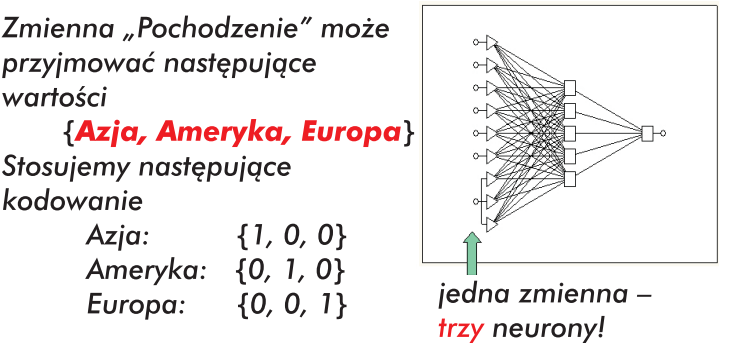
## 2.8. Jak wytłumaczyć sieci, skąd pochodzi krowa?

Kolejny problem jest trudniejszy. Dane dostarczane do sieci neuronowej (lub pozyskiwane z niej na zasadzie odczytania potrzebnego rozwiązania) nie zawsze mają wyłącznie charakter liczbowy – i to stanowi źródło poważnych komplikacji. Niestety, świat jest tak urządzony, że nie wszystko da się zmierzyć i wyrazić ilościowo. Wiele danych, którymi chcemy się posłużyć na wejściu lub na wyjściu sieci neuronowej, ma charakter jakościowy, opisowy, albo – jak to się najczęściej nazywa – *nominalny.* To znaczy ich **wartościami** są pewne **nazwy**, a nie **liczby**. Żeby to wyjaśnić, wygodnie będzie odwołać się do przykładu. Wyobraź sobie, że masz do rozwiązania zadanie, w którym sieć neuronowa rozpoznaje, czy pewne zwierzęta mogą być niebezpieczne dla człowieka, czy nie (niebawem będziesz podobne zadania sam rozwiązywał za pomocą programów, które Ci w tym celu udostępniłem do Twojej zabawy w Internecie).

Otóż zadanie to może wymagać na przykład tego, aby jako daną wejściową podać sieci informację, z jakiej części świata to zwierzę pochodzi. No bo jeśli z podanych danych wejściowych wynika, że zwierzę jest duże, ma rogi i daje mleko, to wiadomo, że jest to krowa. Ale o tym, czy może ona stanowić zagrożenie dla człowieka, decyduje kontynent, na którym to zwierzę spotkamy. Krowy europejskie i azjatyckie są bowiem z reguły spokojne i łagodne, natomiast niektóre krowy amerykańskie, hodowane na otwartych pastwiskach, bywają niebezpieczne. Dlatego w celu rozstrzygnięcia o tym, czy zwierzę może być groźne, czy nie – potrzebna jest wartość zmiennej określającej *pochodzenie* zwierzęcia. I tu pojawia się trudność: wiadomo, jak się nazywają poszczególne kontynenty, ale jak wprowadzić na przykład nazwę *„Azja"* albo *„Ameryka"* na wejście sieci neuronowej?!

Rozwiązaniem postawionego problemu jest użycie reprezentacji nazywanej *„jeden z N"*, gdzie N oznacza liczbę różnych możliwych wartości (nazw),

jakie może przyjmować zmienna nominalna. Sposób kodowania według metody „jeden z N" dla prostego przykładu, w którym N = 3, przedstawiłem na rysunku 2.29. Zasada jest prosta i polega na tym, że dla każdej zmiennej nominalnej, którą chcemy odwzorować na wejściu sieci, umieszcza się w warstwie wejściowej tej sieci tyle neuronów, ile różnych nazw może przybierać ta zmienna (czyli właśnie N). Jeśli na przykład założymy w rozważanym przykładzie, że rozpoznawane zwierzęta mogą pochodzić wyłącznie z Azji, Ameryki lub Europy, to dla reprezentowania na wejściu sieci zmiennej *Pochodzenie* musimy przeznaczyć zestaw trzech neuronów. Jeśli tak zrobimy i potem chcemy powiadomić sieć, że w danym momencie wartością zmiennej *Pochodzenie* jest nazwa *Ameryka,* to podajemy sygnał o wartości 0 na pierwsze z trzech zgrupowanych razem wejść, wartość 1 na drugie z nich oraz ponownie wartość 0 na trzecie wejście.




Rys. 2.29. Sposób kodowania zmiennej nominalnej na **wejściu** sieci neuronowej

Sądzę, że skoro czytasz tę książkę, to jesteś człowiekiem bystrym i pomysłowym, więc na pewno czytając o metodzie „jeden z N" myślałeś sobie w duchu z poczuciem wyższości:

*Po co to tak komplikować? **Ja** mam lepszy pomysł! Zakodujmy sobie, że Azja to 1, Ameryka to 2, a Europa to 3, i wprowadzajmy te wartości na **jedno** wejście sieci! Albo uwzględniając, że sygnał na wejściu sieci powinien przyjmować wartości z przedziału od 0 do 1, może być tak: Azja to 0, Ameryka to ½, a Europa to 1. Dlaczego nikt wcześniej na to nie wpadł?*

Otóż to zaproponowane (hipotetycznie) przez Ciebie rozwiązanie nie jest niestety dobre.

Sieci neuronowe są bardzo wrażliwe na **wzajemne relacje** przedstawianych im wartości. Jeśli przyjmiesz pierwszą regułę kodowania, to sieć neuronowa podczas nauki będzie usiłowała wykorzystać w jakiś sposób fakt (wydedukowany z podawanych jej w zbiorze uczącym danych), że *Europa* to trzy

razy więcej niż *Azja* – i oczywiście wyjdą z tego same bzdury. Jeszcze gorzej będzie w przypadku danych mających charakter przeskalowany, bo wtedy będzie się wydawało, że *Amerykę* da się przeliczyć na *Europę* (mnożąc ją przez 2), ale *Azji* nie da się tak przeskalować, bo zero mnożone przez cokolwiek zawsze pozostaje zerem.

Słowem **musisz** się zgodzić na to, że zmienne nominalne trzeba koniecznie reprezentować techniką „jeden z N", chociaż niestety zwiększa to liczbę wejść w sieci oraz powoduje zwiększenie liczby połączeń między warstwą wejściową a dalszymi warstwami sieci. Zwłaszcza ten drugi fakt jest kłopotliwy, bo musisz pamiętać o tym, że z każdym połączeniem w sieci neuronowej związany jest współczynnik wagowy, którego wartość trzeba potem pracowicie wyznaczyć w trakcie uczenia, więc dodatkowe wejścia (i dodatkowe połączenia) – to dodatkowe kłopoty przy uczeniu. Jednak mimo tych kłopotów nie ma innego (lepszego) rozwiązania i **musisz** (powtórzę to jeszcze raz) zwielokrotniać wejścia związane z każdą zmienną nominalną, stosując schemat „jeden z N" – no chyba że wymyślisz jakąś naprawdę skuteczną i lepszą metodę ich kodowania.

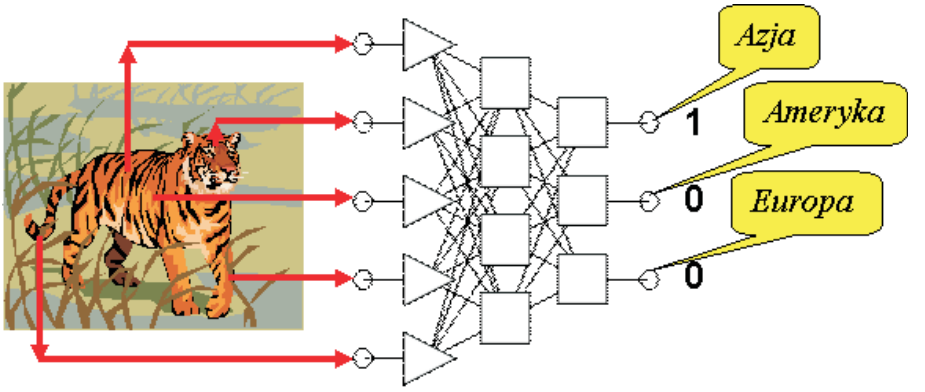
Zanim zakończę ten wątek, dodam tylko jeszcze jedną informację. Otóż zwykle tak jest, że od określonej reguły są także wyjątki. W przypadku reguły „jeden z N" wyjątkiem jest przypadek, kiedy N wynosi dwa. Dla takich „binarnych zmiennych nominalnych", których przykładem może być zmienna *Płeć,* można wyjątkowo zastosować kodowanie polegające na tym, że przyjmiemy (przykładowo) następujące kody: *Mężczyzna = 0* oraz *Kobieta = 1* (lub odwrotnie) – i sieć sobie z tym poradzi. Ale takie binarne dane to w istocie typowy „wyjątek, który potwierdza regułę" – a regułą jest, że dane typu jakościowego **powinno się** kodować metodą „jeden z N".

## 2.9. Jak interpretować odpowiedzi produkowane przez sieć?

Drugą wyróżnioną warstwą, którą omówimy w tym podrozdziale, jest **warstwa wyjściowa**, produkująca ostateczne rozwiązania postawionego problemu. Rozwiązania te są wysyłane na zewnątrz jako **sygnały wyjściowe** z całej sieci, więc ich interpretacja też jest dla nas ważna, bo musimy wiedzieć i rozumieć, o czym nas ta sieć chce powiadomić. Co do ilości neuronów wyjściowych to tu sytuacja jest prostsza niż w przypadku sygnałów wejściowych, bo na ogół wiemy, ile i jakich rozwiązań potrzebujemy i nie mamy tu rozterek typu – dodać dany sygnał czy też go nie dodawać, co zaprzątało naszą uwagę przy projektowaniu warstwy wejściowej sieci. Co do sposobów kodowania to sytuacja na wyjściu sieci jest podobna, jak na jej wejściu, to znaczy zmienne przyjmujące wartości liczbowe trzeba przeskalować tak, żeby wyjściowe neu-

rony mogły wyprodukować tę wartość, która jest prawidłowym rozwiązaniem postawionego problemu, a zmienne nominalne trzeba prezentować na wyjściu korzystając z metody „jeden z N". Jednak na wyjściu sieci można się spodziewać paru problemów specyficznych właśnie dla tej warstwy, dlatego o kilku sprawach warto także i tu podyskutować.

Pierwszy problem szczegółowo związany jest ze stosowaniem metody „jeden z N". Pamiętasz, że przy tej metodzie sygnał o maksymalnej wartości (wynoszącej zwykle 1) może mieć na swoim wyjściu tylko jeden neuron – ten przypisany do właściwej nazwy, będącej wartością zmiennej nominalnej. Wszystkie pozostałe neurony z grupy reprezentującej jedną zmienną powinny mieć wartości równe zero. Tę idealną sytuację wyraźnego wskazania przez sieć określonej nazwy zmiennej nominalnej na wyjściu pokazuje rysunek

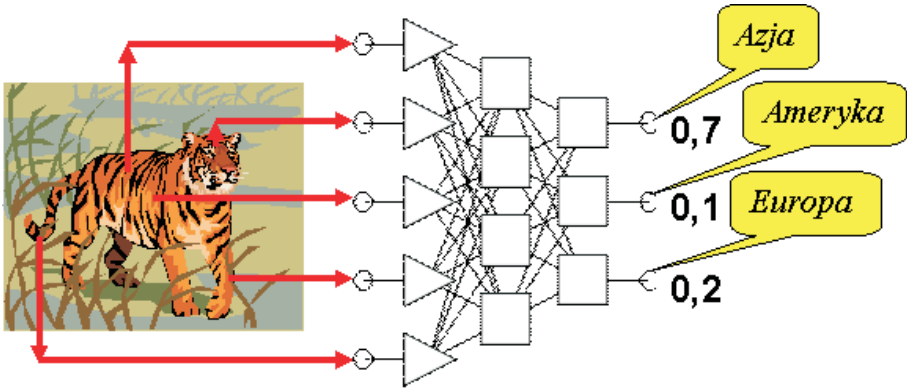



Rys. 2.30. Idealna sytuacja związana z użyciem zmiennej nominalnej na wyjściu sieci neuronowej

2.30. Na rysunku tym przyjęto konwencję, że zadanie przykładowo rozważanej sieci jest (w pewnym sensie) odwróceniem zadania z rysunku 2.29 – tam trzeba było podać **na wejściu**, na jakim kontynencie żyje klasyfikowane zwierzę (żeby dokonać klasyfikacji, czy jest ono niebezpieczne, czy nie), a tutaj **na wejściu** pokazywane[1] jest zwierzę, a sieć ma podać, na jakim kontynencie ono żyje.

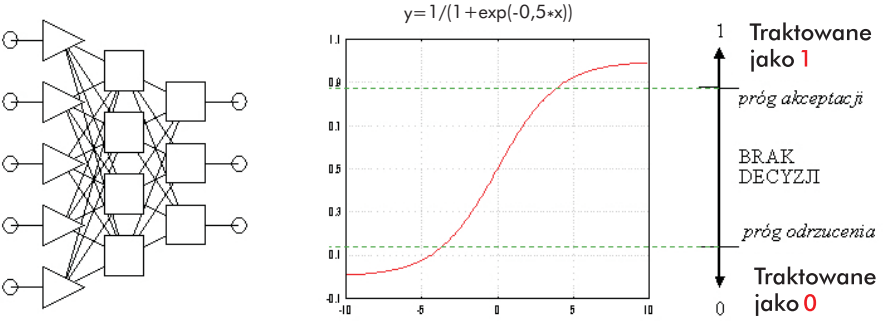
Taka sytuacja jak na rysunku 2.30 zdarza się jednak tylko w teorii (a w praktyce bywa spotykana jedynie jako wynik pomyślnego zbiegu okoliczności). W praktyce ze względu na ograniczoną dokładność pracy sieci, o czym

[1] „Pokazanie" sieci zwierzęcia polega na tym, że do neuronów wejściowej warstwy podawane są wybrane informacje o jego wyglądzie i budowie ciała (jaki ma kształt głowy, jaki kolor futra, jakie łapy, jaki ogon itp.). Symbolizują to czerwone strzałki na rysunku, łączące wybrane elementy budowy zwierzęcia z neuronami wejściowymi, do których dostarczana jest informacja na ich temat.




Rys. 2.31. Rzeczywisty rozkład wartości sygnałów w sieci neuronowej ze zmienną nominalną na wyjściu

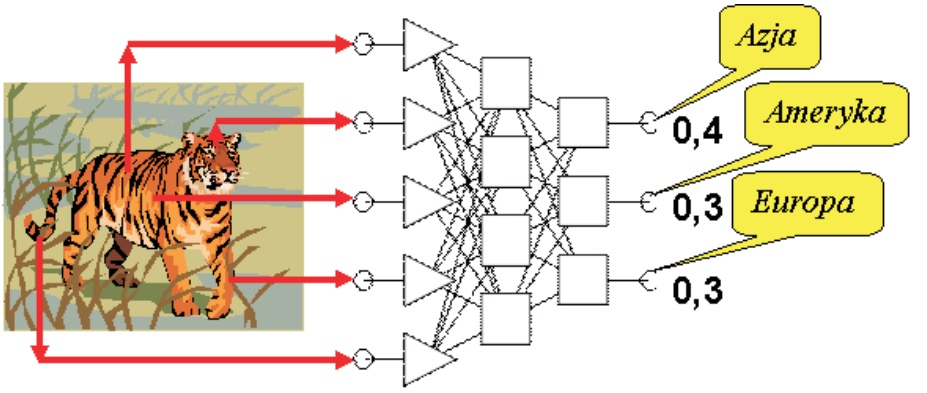
będzie jeszcze niżej mowa, prawie zawsze na wyjściach **wszystkich** neuronów, tworzących grupę przypisaną do jakiejś wyjściowej zmiennej nominalnej, pojawiają się jakieś niezerowe sygnały. Taka sytuacja pokazana jest na rysunku 2.31. No i co z tym zrobić?




Rys. 2.32. Efekt przetwarzania wykańczającego, które powoduje, że wartości wyjściowej zmiennej nominalnej mogą być wyznaczone jednoznacznie, pomimo niedokładnych wartości na wyjściach neuronów

Otóż do uzyskania zdecydowanych odpowiedzi w takich sytuacjach służy przyjęcie dodatkowych kryteriów obróbki wyników dostarczanych przez sieć neuronową w postaci *progu akceptacji* oraz *progu odrzucenia.* Istota tej koncepcji sprowadza się do tak zwanego **przetwarzania wykańczającego** (po angielsku *post-processing*). Sygnały wyjściowe obliczone przez końcowe neurony sieci są zgodnie z tą koncepcją kwantyfikowane poprzez porównanie ich aktualnych wartości ze wspomnianymi progami (patrz rysunek 2.32).

Wartości parametrów, jakimi są próg akceptacji i próg odrzucenia, mogą być dobierane zależnie od potrzeb, podobnie jak reguły rozstrzygania, co należy zrobić, jeśli na jednym lub kilku neuronach pojawi się sygnał braku decyzji. Doświadczenie uczy, że powinno się stawiać sieci **wysokie** wymagania, to znaczy nie próbować „na siłę" doprowadzić do jednoznacznej wartości zmiennej wyjściowej typu nominalnego, na przykład w tak słabo zdeterminowanej sytuacji, jak pokazana przykładowo na rysunku 2.33.



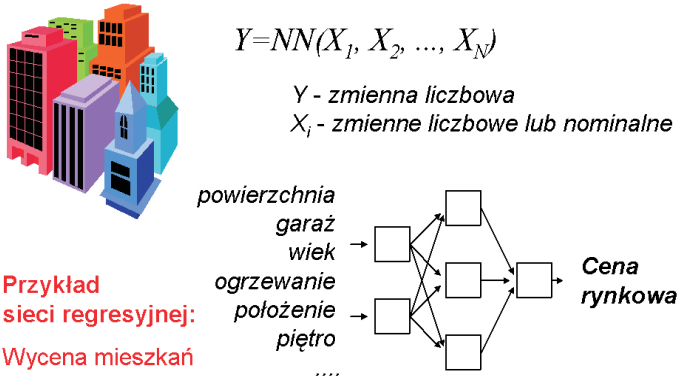

Rys. 2.33. Przykład rozkładu sygnałów wyjściowych, przy którym lepiej jest nie wyznaczać wcale wartości zmiennej nominalnej, niż narazić się na błąd, który jest w tej sytuacji bardzo prawdopodobny

Lepiej zawsze przyznać się do tego, że sieć nie potrafi dokonać jednoznacznej klasyfikacji wejściowego sygnału, niż podejmować konkretną decyzję w warunkach, kiedy z dużym prawdopodobieństwem może to być decyzja błędna.

## 2.10. Co lepiej starać się dostać z sieci – liczbę czy decyzję?

Korzystając z sieci neuronowych pamiętaj zawsze, że wyniki dostarczane przez sieć, nawet jeśli są wartościami liczbowymi (podlegającymi stosownemu skalowaniu), to mają zawsze charakter **przybliżony.** Jakość tego przybliżenia może być różna, jednak o dokładności wielu cyfr znaczących nie może tu być mowy – dobrze, jeśli wynik dostarczany przez neuron ma dokładność lepszą niż dwie cyfry (czyli błąd może dochodzić do kilku procent). Taka jest już po prostu natura tego narzędzia. Świadomość występowania tych ograniczeń zmusza do odpowiedniej **interpretacji** sygnałów wyjściowych, by można było z nich sensownie korzystać, a także skłania do zastanowienia nad tym, jaki model obliczeń neuronowych chcesz wykorzystać. Generalnie

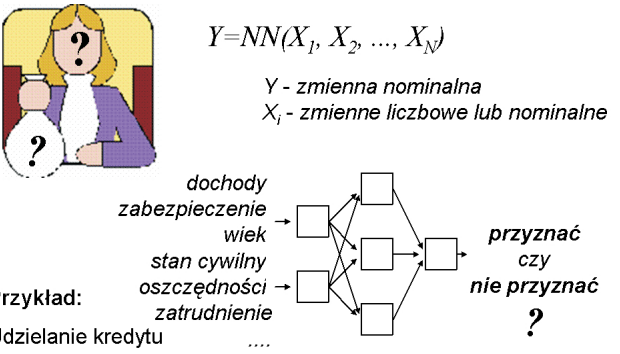
można powiedzieć, że sieci neuronowe mogą tworzyć modele dwóch typów: **regresyjne** oraz **klasyfikacyjne.** Model regresyjny to taki, w którym na wyjściu sieci oczekujemy (i wymagamy) podania konkretnej wartości liczbowej, będącej rozwiązaniem postawionego problemu. Interpretację takiego modelu ilustruje rysunek 2.34.




Rys. 2.34. Przykład sieci realizującej model regresyjny

W modelu podanym na rysunku zadaniem sieci neuronowej jest podanie oszacowania ceny mieszkania. Na wejściu podawane są dane, które mogą mieć charakter liczbowy (na przykład powierzchnia w $m^2$) oraz dane, które mają charakter nominalny (na przykład to, czy z mieszkaniem związany jest garaż), natomiast na wyjściu oczekujemy wartości liczbowej, która określa kwotę, jaka zapewne będzie uzyskana przy sprzedaży mieszkania. Jak wiedzą wszyscy sprzedający lub kupujący mieszkania – ich cena rynkowa jest uzależniona od wielu czynników, przy czym nikt nie potrafi podać ścisłych reguł ekonomicznych, pozwalających z góry przewidzieć, że to mieszkanie będzie kosztowało tyle, a tamto na przykład dwa razy więcej. Pozornie jest to niemożliwe do przewidzenia, bo cena stanowi każdorazowo wynik swobodnej gry rynkowej i suwerennych decyzji podejmowanych przez osoby sprzedające lub kupujące konkretne mieszkanie. A jednak okazuje się, że sieć neuronowa po odpowiednio długiej nauce (na podstawie danych dotyczących wcześniej zawieranych transakcji kupna-sprzedaży mieszkań) potrafi wytworzyć na tyle dobry model regresyjny tego problemu, że rzeczywiste ceny uzyskiwane w kolejnych transakcjach różnią się od prognozy „odgadniętej" przez sieć zaledwie o kilka procent.

Alternatywny model (klasyfikacyjny) związany jest z wymaganiem uzyskania od sieci informacji o zaliczeniu obiektu opisanego przez dane wejściowe do jednej z możliwych klas. Oczywiście, przy tego typu zadaniu na wyjściu sieci umieszczona zostaje zmienna nominalna (patrz rys. 2.35).



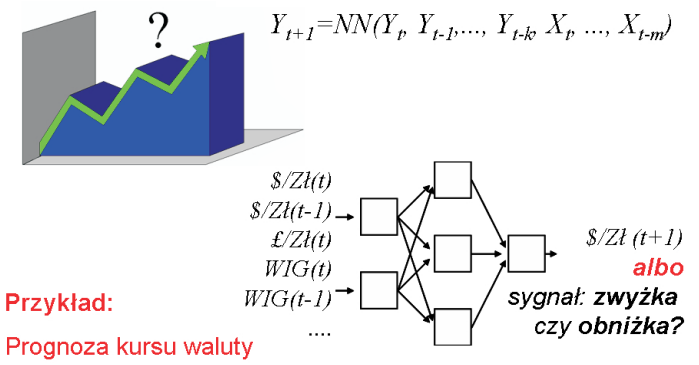

Rys. 2.35. Przykład sieci realizującej model klasyfikacyjny

Pokazany na rysunku przykład dotyczy zadania klasyfikacji osób (albo firm) ubiegających się w banku o kredyt. Takiego klienta urzędnik bankowy musi zaliczyć do jednej z dwóch kategorii: Do pierwszej zalicza się klientów wiarygodnych i odpowiedzialnych, którym warto udzielić kredytu, bo spłacą go na pewno wraz z odsetkami przynosząc w ten sposób bankowi zysk. Do drugiej należą naciągacze i potencjalni bankruci, którzy kredytu nie spłacą, narażając bank na straty. Jak odróżnić jednych od drugich? Ścisłych reguł ani algorytmów nie ma, ale prawidłową decyzję może podpowiedzieć sieć neuronowa, nauczona na podstawie danych z przeszłości. Danych takich jest zwykle sporo, bo każdy bank wcześniej udzielał już wielu kredytów i ma pełne dane o swoich klientach, zarówno tych, którzy spłacili zaciągniętą pożyczkę, jak i tych, którzy dopuścili się defraudacji.

Doświadczenia wielu lat eksploatacji sieci neuronowych dowiodły, że najwygodniej jest tak interpretować stawiane neurokomputerowi zadania, by odpowiedź mogła być udzielona przy zastosowaniu modelu klasyfikacyjnego. Na przykład można domagać się, by sieć określiła, czy zyskowność inwestycji jest *„mała”, „średnia”* lub *„duża”,* względnie czy kredytobiorca jest *„pewny”, „ryzykowny”* lub *„zupełnie niewiarygodny”*. Natomiast wymaganie dokładnego określenia wielkości spodziewanego zysku, dokładności stopnia ryzyka lub wysokości kwoty, jaką można komuś pożyczyć, prowadzić będzie niezawodnie do frustracji, bo sieć na ogół tego zrobić nie potrafi.

**Dlatego liczba wyjść z budowanej sieci bywa często większa niż liczba pytań, na które poszukujemy odpowiedzi.** Dzieje się tak, ponieważ dla wielu sygnałów wyjściowych trzeba sztucznie wprowadzić kilka neuronów obsługujących dane wyjście – na przykład w ten sposób, żeby przewidywany przedział wartości sygnału wyjściowego został podzielony na pewne podzakresy, których rozróżnienie jest dla użytkownika istotne. W takim przypadku nie próbujemy sieci zmuszać do tego, żeby na swoim wyjściu podała konkretną „odgadywaną” wartość, natomiast działamy w taki sposób, że poszcze-

gólne neurony wyjściowe odpowiedzialne są za **sygnalizowanie przynależności aktualnego rozwiązania do określonego przedziału** i to nam na ogół wystarczy**.** Taką sieć klasyfikacyjną znacznie łatwiej zbudować i nauczyć, podczas gdy tworzenie sieci, z której „wyciska się" dokładne rozwiązania problemów matematycznych, jest typową „sztuką dla sztuki" – czasochłonną zabawą o minimalnej przydatności praktycznej. Sformułowane tu wnioski podsumowuje rysunek 2.36, na którym pokazane jest jedno z „klasycznych" zadań, stawianych sieci neuronowej: prognozowanie kursów walut.




Rys. 2.36. Zadanie prognozowania kursów walut jest jednym z wielu zadań, w których twórca sieci neuronowej może wybierać pomiędzy modelem regresyjnym a modelem klasyfikacyjnym

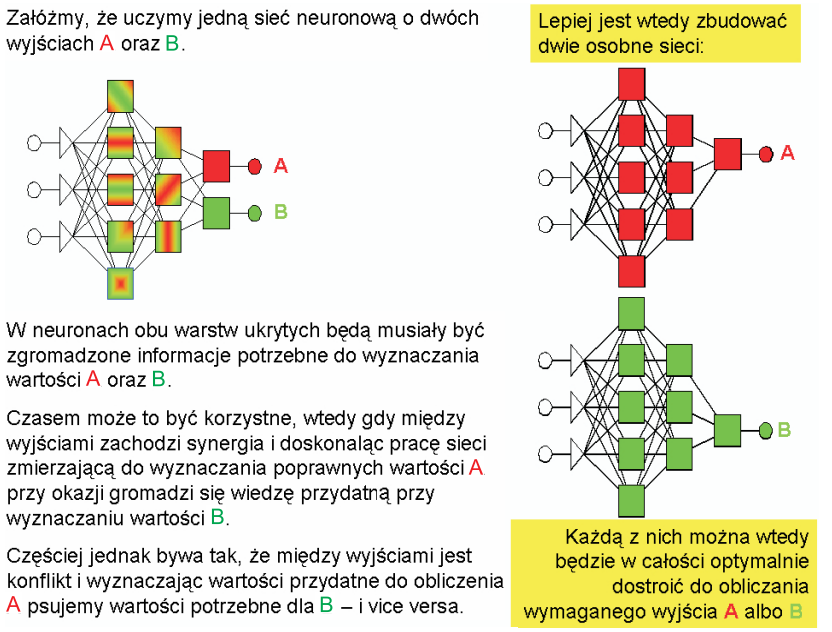
Jak pokazano na rysunku, zadanie to można rozwiązać na dwa sposoby: Albo można budować model prognostyczny, który spróbuje przewidzieć, ile rubli będzie wart dolar w dniu jutrzejszym, albo można się zadowolić modelem, który będzie tylko sygnalizował, czy na drugi dzień należy oczekiwać podwyżki kursu, czy jego obniżenia? To drugie zadanie sieć rozwiąże znacznie łatwiej – a prognoza zwyżki lub obniżki kursu może być użyteczna dla kogoś, kto zamierza na przykład kupić dolary na wyjazd zagraniczny.

## 2.11. Czy lepiej mieć jedną sieć z wieloma wyjściami, czy kilka sieci o jednym wyjściu?

Z warstwą wyjściową sieci neuronowej wiąże się jeszcze jedno zagadnienie, które można rozwiązać na dwa sposoby, a wybór jednego z tych sposobów jest przedmiotem swobodnej decyzji twórcy sieci. Otóż, jak wiadomo, zawsze możliwe jest zbudowanie sieci, która będzie miała dowolną liczbę wyjść – tyle, ile danych wyjściowych chcielibyśmy uzyskać w wyniku rozwiązania postawionego problemu. Nie zawsze jednak jest to optymalne rozwiązanie,

ponieważ proces uczenia sieci o wielu wyjściach musi prowadzić – podczas ustalania wartości wag wewnątrz sieci – do swoistych kompromisów, które zawsze pogarszają wynik.

Kompromisy, o których była mowa, polegają przykładowo na tym, że przy ustalaniu zadań dla pewnego neuronu warstwy ukrytej trzeba uwzględnić (co wyraża się w ustaleniu wartości znajdujących się w nim współczynników wag), jaką rolę będzie ten neuron pełnił w obliczaniu wartości **kilku** neuronów wyjściowych, do których wysyła on swój sygnał wyjściowy, gdy już go obliczy. Może się więc zdarzyć, że rola danego neuronu ukrytego, optymalna z punktu widzenia obliczania z jego pomocą jakiegoś jednego wybranego sygnału wyjściowego z całej sieci, będzie istotnie odmienna niż jego rola optymalna dla któregoś innego wyjścia. W takim przypadku proces uczenia będzie co chwilę zmieniał wartości wag w tym neuronie ukrytym, dostosowując go raz do jednej, a raz do drugiej roli – z oczywistym skutkiem w postaci długiego i mało skutecznego uczenia. Dlatego często lepiej jest podzielić złożony problem i zamiast jednej sieci o wielu wyjściach zbudować kilka oddzielnych sieci, które wykorzystują ten sam zestaw danych wejściowych, ale mają rozdzielone warstwy ukryte oraz pojedyncze wyjścia (rys. 2.37).

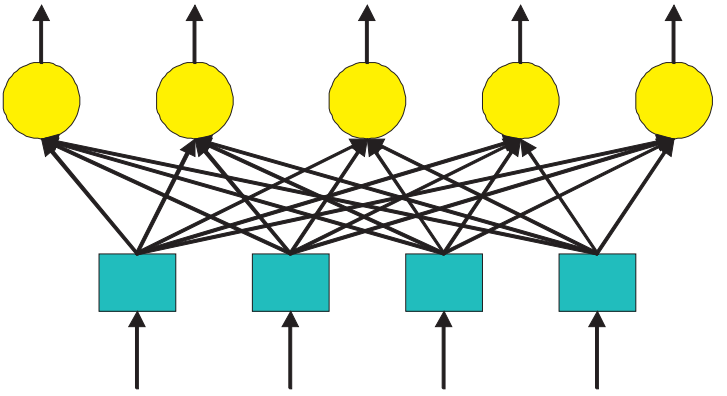



Rys. 2.37. W jednej sieci o wielu wyjściach neurony ukryte muszą tak się uczyć, żeby obsłużyć obydwa wyjścia. Czasem zamiast stosować jedną sieć o wielu wyjściach, lepiej jest zastosować kilka oddzielnych sieci mających te same sygnały wejściowe, ale każdorazowo tylko jedno wyjście, bo neurony ukryte mogą się specjalizować

Reguły zasugerowanej w związku rysunkiem 2.37 nie należy jednak traktować zbyt dogmatycznie, gdyż czasem zdarza się, że sieć o wielu wyjściach uczy się lepiej od tej o jednym wyjściu. Ten paradoksalny na pozór wynik można uzasadnić faktem, że opisany w poprzednim akapicie potencjalny „konflikt" związany z funkcjonowaniem neuronów ukrytych i z ustalaniem ich roli podczas wypracowywania kilku różnych wartości wyjściowych może wcale nie wystąpić. Przeciwnie, niekiedy obserwuje się przy uczeniu sieci efekt swoistej synergii, polegający na tym, że ustalając (w toku procesu uczenia) parametry neuronu ukrytego, optymalne z punktu widzenia „interesów" kilku neuronów warstwy wyjściowej, osiąga się sukces szybciej i skuteczniej, niż gdy się to robi w oddzielnych sieciach, indywidualnie dostrajanych dla każdego sygnału wyjściowego osobno. Dlatego nie trzeba się z góry nastawiać na to, że z pewnością lepsze jest jedno albo drugie rozwiązanie, tylko korzystne jest wypróbowanie obydwu i wybranie lepszego z nich. Moja własna obserwacja, wynikająca z tego, że sam zbudowałem już setki sieci do różnych zastosowań, a także konsultowałem dziesiątki prac moich studentów i doktorantów, wskazuje na to, że **znacząco częściej** optymalnym rozwiązaniem jest jednak kolekcja sieci o pojedynczych wyjściach – chociaż biologiczne sieci neuronowe są częściej zorganizowane na zasadzie agregatów wielowyjściowych.

Wiesz już, że każda sieć *feedforward* musi mieć przynajmniej dwie wymienione wyżej warstwy – wejściową i wyjściową. Bywają zresztą takie sieci (rys. 2.38) – nazywa się je wtedy sieciami **jednowarstwowymi,** bo mają tylko jedną warstwę uczącą się. Jest to oczywiście warstwa wyjściowa, bo warstwa wejściowa, jak już wiesz, w żadnej sieci nie podlega uczeniu.

Jednak wiele sieci (zwłaszcza tych rozwiązujących bardziej złożone zadania) musi dysponować dodatkowymi warstwami elementów, pośredniczących



Rys. 2.38. Przykład sieci jednowarstwowej

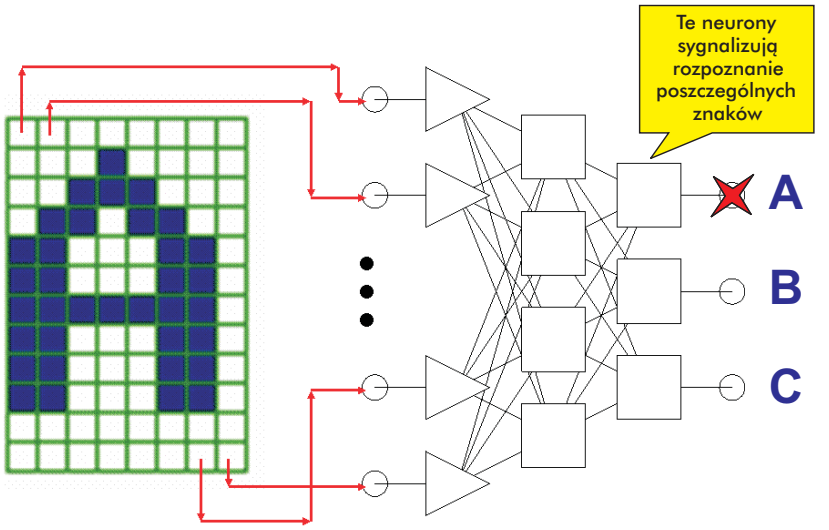
pomiędzy wejściem i wyjściem. Te warstwy wprowadzone pomiędzy wejście i wyjście nazywane są zwykle **warstwami ukrytymi**. Nazwa ta brzmi niezwykle i tajemniczo, dlatego możesz się czuć zakłopotany, stykając się z problemem warstw ukrytych po raz pierwszy. Chciałbym Ci w związku z tym jak najszybciej wyjaśnić, czym są te warstwy i w jakim sensie są one „ukryte".

## 2.12. Co się kryje w warstwach ukrytych?

Najkrócej można powiedzieć, że warstwy ukryte stanowią narzędzie, służące do takiego przetworzenia sygnałów wejściowych (odebranych przez warstwę wejściową), by warstwa wyjściowa mogła łatwiej znaleźć potrzebną odpowiedź (rozwiązanie problemu). Działanie tych dodatkowych (nie zawsze występujących) warstw nie jest możliwe do bezpośredniej obserwacji przez użytkownika, który stawia sieci zadania i obserwuje, czy sieć poprawnie je wykonuje. Nie ma on dostępu ani do wejść neuronów tych warstw (przy przesyłaniu do nich sygnałów trzeba korzystać z pośrednictwa neuronów warstwy wejściowej), ani do ich wyjść (efekty działania neuronów warstw ukrytych ujawniają się wyłącznie pośrednio, poprzez odpowiedzi, jakie produkują i wysyłają neurony warstwy wyjściowej). Jak z tego wynika – działanie neuronów warstw pośredniczących nie jest bezpośrednio widoczne dla użytkownika sieci – i w tym właśnie sensie są one *ukryte*.

Chociaż ukryte – te dodatkowe warstwy w sieciach neuronowych pełnią bardzo ważną i odpowiedzialną funkcję. Tworzą one dodatkową strukturę przetwarzającą informację, a ich rolę najłatwiej jest przedyskutować na przykładzie sieci realizujących często pojawiające się zadanie **rozpoznawania obrazów.** W sieciach takich na wejście (do pierwszej warstwy sieci) podaje się obraz cyfrowy. Nie może to być obraz z typowego aparatu fotograficznego ani zwykły tradycyjny obraz wczytany do systemu przez skaner lub *Frame Grabber*. Powód jest bardzo prosty: w takim systemie zasilanym na wejściu samym obrazem, jakim takim, a nie jakimś jego bardziej wyrafinowanym opisem (porównaj rysunek 2.30) wejściowa warstwa neuronów odpowiada swymi rozmiarami i organizacją (zgrupowaniem neuronów w odpowiednie wiersze i kolumny) organizacji samego obrazu. Po prostu do każdego punktu obrazu przypisany jest neuron wejściowy sieci, który analizuje i sygnalizuje jego stan. Obraz z cyfrowego aparatu fotograficznego lub ze skanera ma postać zbioru pikseli, ale tych pikseli są **miliony.** Niepodobna zbudować sieć, która by na wejściu miała miliony neuronów wejściowych, zwłaszcza że taka sieć musiałaby mieć miliardy (!) połączeń, dla których trzeba byłoby wyznaczać wartości wag. Taka konstrukcja jest całkowicie niemożliwa do realizacji.

Można jednak sobie wyobrazić system, który na wejściu będzie otrzymywał obraz cyfrowy bardzo uproszczony, złożony z niewielkiej liczby pikseli, układających się w formę nieskomplikowanych znaków – na przykład liter (rys. 2.39).




Rys. 2.39. Przykładowa sieć rozpoznająca proste obrazy

Na wyjściu takiej sieci oczekuje się decyzji informujących o tym, co rozpoznano – na przykład może być ona tak zorganizowana, że do poszczególnych neuronów wyjściowych przypiszemy umownie pewne decyzje – na przykład *„rozpoznano literę A”, „rozpoznano literę B”* itp., a wielkości sygnałów na tych wyjściach interpretować się będzie w kategoriach stopnia pewności odpowiedniej decyzji. Łatwo zauważyć, że możliwe jest w związku z tym podawanie przez sieć odpowiedzi wieloznacznych (*„to coś jest podobne w stopniu 0,7 do litery A, ale w stopniu 0,4 przypomina także literę B”*). Jest to jedna z ciekawych i użytecznych cech sieci neuronowych, które można w związku z tym kojarzyć z systemami o rozmytej (*fuzzy*) logice działania – obszerniejsze omówienie tego aspektu wykracza jednak poza tę książkę.

Neurony warstwy ukrytej pełnią w omawianej tu sieci rolę **pośredników** – mają one bezpośredni dostęp do danych wejściowych, czyli oglądają pokazany sieci obraz, a na podstawie ich wyjść dalsze warstwy podejmują określone decyzje o rozpoznaniu takiego lub innego obrazu. Dlatego uważa się, że rola neuronów warstwy ukrytej polega na tym, by wypracowywały zestawy takich **wstępnie przetworzonych** danych wejściowych, z których korzystać będą neurony warstwy wyjściowej przy określaniu końcowego wyniku.

Przydatność tych warstw pośrednich wynika z faktu, że na ogół dokonanie pewnych przekształceń danych wejściowych sprawia, że rozwiązanie stawianego przed siecią zadania staje się znacznie łatwiejsze niż w przypadku próby rozwiązywania zadania w sposób bezpośredni. Na przykład, w rozważanym tu zadaniu rozpoznawania obrazów trudno jest znaleźć regułę pozwalającą ustalić, jaki obiekt pokazano, gdy analizuje się wyłącznie same jasne i ciemne piksele obrazu (a tylko taką informacją dysponuje sieć bez warstwy ukrytej). Znalezienie prawidłowej reguły rozpoznawania funkcjonującej na tak niskim poziomie jest trudne, ponieważ rozpoznawane obiekty mogą występować w wielu odmianach różniących się nieco kształtem, ale mających to samo znaczenie. Ten sam obiekt (na przykład litera **A**) może bowiem wyglądać trochę inaczej, gdy jest skanowany z rękopisu, inaczej gdy pochodzi z tekstu drukowanego, a jeszcze inaczej, gdy został na przykład sfotografowany na powiewającym transparencie. Nawet litera wydrukowana dokładnie tą samą czcionką może być reprezentowana przez dwa obrazy mające zupełnie inne wartości pikseli w pewnych wybranych punktach, jeśli jest na przykład przesunięty (takie banalne zakłócenie, w niczym nie przeszkadzające człowiekowi, który czyta tekst, może zmienić obraz cyfrowy litery w zupełnie dramatyczny sposób, jeśli musimy przywiązywać wagę do tego, który konkretnie piksel jest biały, a który czarny). Jakby tego było mało – przy obrazach cyfrowych o małej rozdzielczości pojawia się dodatkowa trudność polegająca na tym, że zupełnie różne obiekty mogą mieć na cyfrowych obrazach bardzo duże zbiory identycznych pikseli.

Oczekiwanie, że sieć neuronowa „jednym skokiem" pokona wszystkie te trudności, jakie dzielą „surowy" obraz od finalnej decyzji dotyczącej rozpoznania tego, co ten obraz przedstawia – byłoby w takim przypadku mało realistyczne. Rozumowanie to jest prawidłowe, gdyż odpowiednie doświadczenie potwierdza, że żaden proces uczenia nie zdoła zmusić prostej sieci bez warstwy ukrytej do tego, by potrafiła to zadanie rozwiązać. Taka sieć bez warstw ukrytych musiałaby sobie wytworzyć jakiś taki zdumiewający mechanizm, dzięki któremu raz rozumiałaby pewne wybrane i ustalone zestawy pikseli jako należące do różnych obiektów, a innym razem kojarzyłaby różne zestawy pikseli z tym samym obiektem. Tego się po prostu zrobić nie da.

To, czego nie może zrobić sieć o mniejszej liczbie warstw, potrafi na ogół wykonać sieć zawierająca warstwę ukrytą. Neurony tej warstwy będą w takich zadaniach znajdowały pewne wartości pomocnicze, znacząco ułatwiające rozwiązanie postawionego zadania. Na przykład, w rozważanym zadaniu rozpoznawania obrazów neurony warstwy ukrytej mogą wykrywać i kodować ogólne **cechy** opisujące strukturę obrazu i widocznych na nim obiektów. Cechy te powinny lepiej odpowiadać wymaganiom związanym z ostatecznym

rozpoznawaniem obrazu niż sam oryginalny obraz – na przykład mogą być w pewnym stopniu niezależne od położenia czy skali rozpoznawanych obiektów. Przykładem cech opisujących obiekty na obrazach liter (ułatwiających ich rozpoznawanie) mogą być:

- obecność lub brak w rysunku litery zamkniętych konturów (posiadają je litery O, D, A i kilka innych, nie mają tej cechy litery I, T, S, N itd.),
- czy znak ma wcięcie u dołu (A, X, K) albo u góry (U, Y, V, K), a może z boku (E, F),
- czy ma kształt raczej zaokrąglony (O, S, C), czy ostry (E, W, T, A).

Oczywiście cech różnicujących litery i równocześnie niewrażliwych na typowe czynniki utrudniające ich rozpoznawanie (zmiana kroju czcionki lub pisma ręcznego, zmiana wielkości litery, jej pochylenie, przesunięcie itd.) można znaleźć znacznie więcej. Należy jednak z naciskiem podkreślić, że twórca sieci neuronowej nie musi sam jawnie zadawać, jakie to cechy obrazu mają być znajdowane, gdyż sieć może sama nabywać stosownych umiejętności w trakcie procesu uczenia. Oczywiście, nie ma żadnej gwarancji, że sieć neuronowa przeznaczona do rozpoznawania liter nauczy swoją warstwę ukrytą wykrywania i sygnalizowania właśnie tych cech, które wyżej wymieniłem. Można nawet się założyć o dużą sumę, że tych właśnie, przeze mnie wskazanych cech różnicujących litery, sieć neuronowa sama nie wykryje. Jednak doświadczenia niezliczonych badaczy potwierdzają, że sieci neuronowe w zdaniach **rozpoznawania** potrafią znajdować cechy, które **skutecznie** ułatwiają rozpoznawanie – chociaż na ogół twórca sieci nie potrafi zrozumieć, jaki sens ma taka lub inna cecha kodowana w warstwie ukrytej.

Użycie warstwy ukrytej jest więc ze wszech miar celowe, bo sieć neuronowa może dzięki niej przejawiać znacznie inteligentniejsze formy zachowania, trzeba jej jednak dać szansę poprzez uwzględnienie w strukturze elementów, które mogą właśnie takie „opisowe cechy obrazu" wydobywać. Warto dodać, że charakter wydobywanych cech nie jest zdeterminowany przez strukturę sieci, tylko przez proces uczenia. Jeśli wyobrazimy sobie sieć, która ma rozpoznawać obrazy, to cechy, które będą wydobywały neurony warstwy ukrytej, dostosują się automatycznie do rodzaju rozpoznawanych obrazów. Jeśli nakażemy sieci wykrywanie na zdjęciach lotniczych obrazów zamaskowanych wyrzutni rakietowych, to staje się oczywiste, że głównym zadaniem warstwy ukrytej będzie uniezależnienie się od położenia obiektu, gdyż podejrzany kształt może się pojawić w dowolnym rejonie obrazu i zawsze powinien być tak samo rozpoznany. Jeśli natomiast sieć ma rozpoznawać litery, to nie powinna gubić informacji o ich położeniu (mało przydatna będzie sieć, która poinformuje nas, że gdzieś tam na zeskanowanej stronicy znajduje się litera A – my musimy wiedzieć, gdzie ona jest, a dokładniej – w jakim kontekście

występuje), więc w tym przypadku celem warstwy ukrytej może być wydobycie cech pozwalających na niezawodne rozpoznanie litery niezależnie od jej rozmiaru i niezależnie od kroju czcionki (font). Co ciekawe – oba te zadania może rozwiązywać (po odpowiednim treningu) ta sama sieć – chociaż oczywiście sieć nauczona rozpoznawać czołgi nie potrafi odczytywać pisma, a nauczona identyfikować odciski palców nie poradzi sobie z rozpoznawaniem twarzy.

## 2.13. Ile potrzeba neuronów, żeby otrzymać dobrze działającą sieć?

Jak wynika z przytoczonych uwag – najszersze możliwości zastosowań mają sieci posiadające przynajmniej trójwarstwową strukturę, z wyróżnioną warstwą wejściową przyjmującą sygnały, warstwą ukrytą wydobywającą potrzebne cechy wejściowych sygnałów oraz z warstwą wyjściową podejmującą ostateczne decyzje i podającą rozwiązanie. W tej strukturze pewne elementy są zdeterminowane: liczba elementów wejściowych i wyjściowych, a także zasada połączeń (każdy z każdym) pomiędzy kolejnymi warstwami. Są jednak elementy zmienne, które trzeba wybrać samemu: jest to **liczba warstw ukrytych** (jedna czy kilka?) oraz **liczba elementów w warstwie** (warstwach?) ukrytych (rys. 2.40).


Rys. 2.40. Najbardziej istotny problem przy wyborze struktury sieci neuronowej wiąże się z ustaleniem, ile elementów powinna mieć warstwa ukryta?

Ponieważ mimo wielu lat rozwoju tej techniki wciąż nie udało się stworzyć precyzyjnej teorii sieci neuronowych – elementy te wybiera się zwykle arbitralnie albo metodą prób i błędów. Zdarza się, że pomysł twórcy sieci na

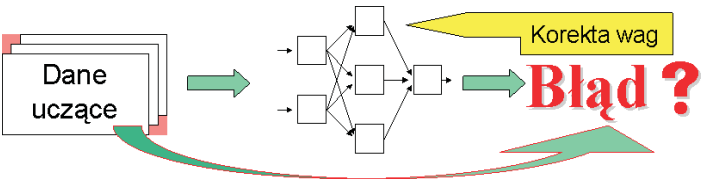
temat tego, ile neuronów ukrytych należy zastosować, a także jak je rozmieścić (np. w postaci jednej warstwy ukrytej albo w postaci kilku takich warstw), nie jest najbardziej trafny. Jednak nie powinno to mieć **zasadniczego** wpływu na sposób działania sieci, gdyż w trakcie procesu uczenia ma ona zawsze możliwość skorygowania ewentualnych błędów struktury poprzez wybór odpowiednich parametrów połączeń. Trzeba jednak wyraźnie przestrzec przed dwoma rodzajami błędów, stanowiących pułapki dla wielu (zwłaszcza początkujących) badaczy sieci neuronowych.

Pierwszy błąd polega na zaprojektowaniu sieci o zbyt małej liczbie elementów – jeśli warstwy ukrytej nie ma wcale lub gdy występuje w niej zbyt mało neuronów, proces uczenia może się definitywnie nie udać, gdyż sieć nie ma szans odwzorować w swojej (zbyt ubogiej) strukturze wszystkich szczegółów i niuansów rozwiązywanego zadania. Pokażę Ci potem konkretne przykłady ilustrujące fakt, że za mała i zbyt prymitywna sieć nie potrafi rozwiązać pewnych zadań – nawet jeśli się ją uczy bardzo długo i bardzo dokładnie. Po prostu z sieciami neuronowymi bywa tak jak z ludźmi: nie wszystkie sieci są wystarczająco uzdolnione, żeby rozwiązać określone zadanie. Na szczęście możesz zawsze bardzo łatwo poznać, czy sieć jest bardziej, czy mniej inteligentna, bo widzisz jej strukturę i możesz policzyć neurony, a miarą zdolności sieci jest tu po prostu liczba neuronów ukrytych. U ludzi trudniej to rozpoznać!

Niestety, mimo swobody budowania większych albo mniejszych sieci zdarza się czasem, że ktoś zaprojektuje sieć o zbyt małej inteligencji. Skutkuje to zawsze niepowodzeniem przy próbie zastosowania takiej sieci do jakiegoś konkretnego celu, bo taki „neuronowy głupek" mający zbyt mało neuronów ukrytych nigdy nie rozwiąże stawianych mu zadań, niezależnie od tego, jak bardzo będziesz się trudził, usiłując go czegoś nauczyć.

Niestety z inteligencją sieci nie można także „przedobrzyć". Mówiąc obrazowo, efektem **nadmiernej** inteligencji sieci wcale nie jest większa sprawność w rozwiązywaniu stawianych jej zadań, tylko zdumiewający efekt polegający na tym, że sieć zamiast pracowicie zdobywać użyteczną wiedzę – zaczyna oszukiwać nauczyciela i w efekcie wcale się nie uczy! Brzmi to w pierwszej chwili wręcz nieprawdopodobnie, ale taka jest prawda. Sieć, która ma zbyt wiele warstw ukrytych lub zbyt wiele elementów w warstwach ukrytych, ma skłonność do upraszczania sobie zadania i w efekcie jeśli tylko może, „idzie na łatwiznę". Żebyś zrozumiał, na czym to polega, muszę Ci w kilku słowach powiedzieć, jak przebiega proces uczenia sieci (rys. 2.41).

Szczegóły tego procesu poznasz w jednym z dalszych rozdziałów, ale już teraz muszę Ci powiedzieć, że sieć naucza się podając jej na wejście sygnały, dla których znane są prawidłowe rozwiązania, bo są one zawarte w **danych**




Rys. 2.41. Maksymalnie uproszczony schemat procesu uczenia sieci neuronowej

**uczących**. Dla każdego podanego zestawu danych wejściowych sieć usiłuje podać na wyjściu swoją propozycję rozwiązania. Na ogół rozwiązania wyznaczone przez sieć są inne niż prawidłowe rozwiązania podane w danych uczących, więc po skonfrontowaniu tego rozwiązania, które podaje sieć, z tym rozwiązaniem, które znajduje się w danych uczących jako **wzorzec poprawnego rozwiązania –** ujawniane jest, jak duży błąd sieć popełniła w swoim rozwiązaniu. Na podstawie oceny błędu algorytm nauczający sieć zmienia wagi wszystkich jej neuronów w taki sposób, żeby na przyszłość sieć już nie popełniała takich samych błędów.

Naszkicowany wyżej schemat procesu uczenia wskazuje, że sieć dąży do tego, **żeby nie popełniać błędów podczas prezentowania jej danych uczących.** Dobrze ucząca się sieć szuka w związku z tym takiej reguły przetwarzania sygnałów wejściowych, by w następstwie jej zastosowania nauczyć się zasady obliczania poprawnych rozwiązań. Jeśli sieć odkryje taką regułę, to potem potrafi rozwiązywać zarówno te zadania, które jej pokazywano w ramach danych uczących, jak i inne, podobne zadania, które będą jej przedstawiane w trakcie normalnej eksploatacji. Mówimy wtedy, że sieć wykazuje zdolność do uczenia się oraz do **generalizacji** (uogólniania) wyników uczenia i traktujemy to jako sukces.

Niestety, nadmiernie inteligentna sieć, mająca pokaźne zasoby pamięciowe w postaci dużej liczby neuronów ukrytych (wraz z ich nastawialnymi zestawami wag), może łatwo uniknąć popełniania błędów w trakcie procesu uczenia w taki sposób, że nauczy się całego zbioru danych uczących na pamięć. Wtedy niesłychanie szybko odnosi duże sukcesy w procesie uczenia, ponieważ na każde zadane jej pytanie zna i podaje prawidłową odpowiedź. Niestety, przy tym sposobie „fotografowania" danych uczących sieć ucząc się na podstawie przedstawianych jej przykładów poprawnych rozwiązań stawianych jej zadań nie podejmuje próby uogólniania nabywanych wiadomości, tylko usiłuje osiągać sukces na zasadzie dokładnego zapamiętania reguł w rodzaju „przy takim wejściu takie wyjście".

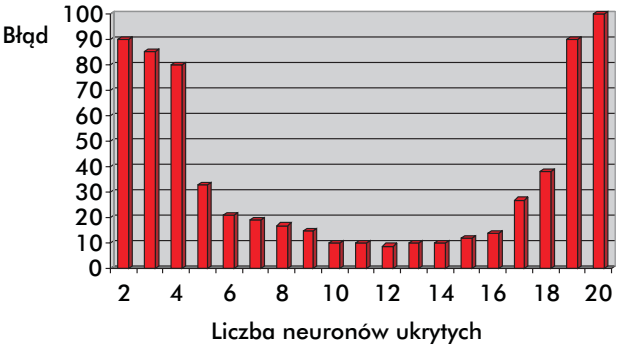
Objawem takiego nieprawidłowego działania sieci jest fakt, że uczy się ona dokładnie i szybko całego tzw. ciągu uczącego (czyli zbioru przykładów

użytych do pokazania sieci, jak należy rozwiązywać stawiane zadania), natomiast fatalnie kompromituje się przy pierwszej próbie **egzaminu**, czyli rozwiązania zadania z podobnej klasy, ale jednak nieco odmiennego od zadań jawnie pokazywanych w trakcie uczenia. Przykładowo ucząc sieć rozpoznawania liter bardzo szybko uzyskujemy sukces (sieć bezbłędnie rozpoznaje wszystkie pokazywane jej litery), ale próba pokazania jej litery napisanej odmiennym charakterem pisma lub wydrukowanej innym fontem – prowadzi do zupełnego braku rozpoznania (na wszystkich wyjściach sieci są zera) lub do rozpoznań ewidentnie błędnych. Bliższa analiza wiedzy zgromadzonej przez sieć ujawnia w takich przypadkach, że zapamiętała ona wiele prostych reguł w rodzaju „jak tutaj są dwa piksele zapalone, a tam jest pięć zer – to należy rozpoznać literę A". Te prymitywne reguły nie wytrzymują oczywiście konfrontacji z nowym zadaniem i sieć zawodzi nasze oczekiwania.

Opisany objaw „uczenia się na pamięć" nie występuje w sieciach mających mniejszą warstwę ukrytą, gdyż dysponując ograniczoną pamięcią sieć taka musi lepiej się starać i wypracować na niewielu dostępnych jej elementach warstwy ukrytej takie reguły przekształcania wejściowego sygnału, by umożliwić jego trafne wykorzystanie w więcej niż jednym przypadku wymaganej odpowiedzi systemu. Z reguły prowadzi to do znacznie wolniejszego i bardziej uciążliwego procesu uczenia (przykłady, na podstawie których sieć ma się uczyć, muszą być pokazane więcej razy – często kilkaset lub kilka tysięcy razy), jednak efekt końcowy jest zwykle znacznie lepszy. Po zakończeniu prawidłowo prowadzonego procesu uczenia z chwilą stwierdzenia dobrego działania sieci dla przykładów stanowiących podstawę uczenia mamy prawo przypuszczać, że poradzi sobie ona także z podobnymi (chociaż nie identycznymi) zadaniami, pokazanymi jej w trakcie egzaminu. Nie zawsze tak jest, ale zwykle tak **bywa** i na tym musimy opierać swoje oczekiwania dotyczące użycia sieci.

Zapamiętajmy więc na koniec następującą regułę: Nie należy oczekiwać cudów, dlatego przypuszczenie, że mało skomplikowana sieć, mająca niewiele ukrytych neuronów, rozwiąże skomplikowane zadanie, jest raczej przypuszczeniem mało realistycznym. Jednak zastosowanie zbyt wielu warstw ukrytych lub zbyt wielu neuronów prowadzi także do znacznego pogorszenia sprawności procesu uczenia. Gdzieś pomiędzy tymi skrajnościami jest zlokalizowana optymalna wielkość warstwy ukrytej.

Rysunek 2.42 pokazujący (na podstawie konkretnych symulacji komputerowych), jak wygląda błąd popełniany przez sieć przy różnej liczbie neuronów ukrytych, potwierdza, że jest wiele sieci, które działają prawie tak samo dobrze, a mają różną liczbę neuronów ukrytych – więc w to obszerne optimum nie tak trudno jest się „wstrzelać". Jednak skrajności (to znaczy zbyt dużych i zbyt małych sieci) należy unikać. Szczególnie szkodliwe jest




Rys. 2.42. Przykładowa zależność błędu popełnianego przez sieć od liczby neuronów ukrytych

użycie dodatkowych (nadmiarowych) warstw ukrytych i nie należy się specjalnie dziwić, że często lepsze wyniki daje sieć o mniejszej liczbie warstw ukrytych (bo można ją porządnie nauczyć) niż – teoretycznie lepsza – sieć z większą liczbą warstw ukrytych, w której jednak proces uczenia „grzęźnie" w nadmiarze szczegółów. Dlatego należy stosować sieci o jednej lub (wyjątkowo!) dwóch warstwach ukrytych, natomiast pokusę stosowania sieci o większej liczbie warstw ukrytych najlepiej przezwyciężać stosując post i zimne kąpiele.

## 2.14. Pytania kontrolne i zadania do samodzielnego wykonania

1. Wymień kilka najbardziej istotnych różnic, jakie występują pomiędzy sztucznymi sieciami neuronowymi a rzeczywistymi biologicznymi strukturami, dającymi się zidentyfikować w mózgu człowieka i zwierząt. Spróbuj odpowiedzieć, które z tych różnic są wynikiem **kapitulacji** (nie udaje się nam zbudować dostatecznie skomplikowanego sztucznego neuronu, więc zadowalamy się namiastką), a które są wynikiem świadomego, celowego wyboru twórców sieci neuronowych?

2. Omów pojęcie **wagi synaptycznej** od strony roli, jaką odgrywa w sztucznych sieciach neuronowych oraz od strony właściwości biologicznych, które opisuje. W układach elektronicznych modelujących sieci neuronowe wagi są niekiedy **dyskretne**, to znaczy mogą przyjmować wyłącznie niektóre wartości (na przykład całkowito-liczbowe) – a nie całkiem dowolne. Jak sądzisz, jaki to ma wpływ na działanie sieci?

3. Przedstaw schemat przetwarzania informacji wejściowych w sygnał wyjściowych, zachodzący w sztucznym neuronie oraz wskaż, czym mogą się

różnić od siebie poszczególne warianty sztucznych neuronów (liniowe, MLP, RBF).

4. Omów krok po kroku działanie sieci zbudowanej ze sztucznych neuronów. Dlaczego warstwa ukryta nosi taką właśnie nazwę?

5. Omów problem związku **struktury** sieci neuronowej i wykonywanych przez nią **funkcji**. Jakie bywają najczęściej spotykane struktury sieci i jakie są ich właściwości? Jak wyjaśnić możliwość wypełniania sensownych zadań obliczeniowych przez sieć, która ma losowe (przypadkowe) połączenia elementów?

6. Jak sądzisz, o czym świadczy fakt, że sieć neuronowa wykazuje dużą odporność na uszkodzenia jej elementów oraz całej struktury?

7. Co powoduje, że w sieciach neuronowych tak radykalnie trzeba odróżniać dane typu **ilościowego** od danych typu **jakościowego** (danych nominalnych)? Istnieje opinia, że sieć, w której wykorzystuje się na wejściu lub na wyjściu dane nominalne, wymaga dłuższego uczenia, ponieważ jest w niej więcej połączeń, których wagi trzeba ustalić. Czy jest to pogląd uzasadniony?

8. Sieć która ma na wyjściu dane kodowane metodą „*jeden z N*" może czasem odmawiać udzielenia odpowiedzi na postawione jej pytanie. Jaki może być skutek tego faktu? Czy wskazana okoliczność powinna być rozumiana jako zaleta, czy jako wada takiej sieci?

9. Czy zadanie analizy serii czasowych i związanej z tym prognozy (patrz rys 2.36) można uznać za bardziej podobne do zadania regresyjnego (rys. 2.34), czy do zadania klasyfikacyjnego (rys. 2.35)? Jakie argumenty przemawiają za jedną, a jakie za drugą decyzją?

10. Czy z rozważań przedstawionych na rysunku 2.37 (oraz w tekście powiązanym z tym rysunkiem) należy wyciągać wniosek, że zawsze niekorzystne jest stosowanie sieci neuronowej o wielu wyjściach? W szczególności czy mając problem, w którym na wyjściu sieci oczekujemy rozwiązania w postaci zmiennej nominalnej, kodowanej techniką „jeden z N", można zamiast jednej sieci o N wyjściach zastosować N sieci, z których każda będzie miała jedno wyjście?

11. **Zadanie dla zaawansowanych:** Wyobraźmy sobie, że w pewnym zadaniu klasyfikacyjnym, które chcemy rozwiązać z użyciem sieci neuronowych, mamy do dyspozycji pewną liczbę sygnałów wejściowych, co do których istnieją wątpliwości, czy wnoszą one jakieś wartościowe informacje w kontekście rozważanej klasyfikacji, czy też nie. Na przykład, czy wynik pewnej analizy krwi pozwala zaliczyć pacjenta do grupy osób dobrze znoszących proponowane leczenie, czy przeciwnie – ten wynik niczego w przedmiotowej sprawie nie wyjaśnia. Otóż mając takie wątpliwe dane wejściowe, możemy przeżywać rozterki, czy należy je podawać na wejście sieci, bo naj-

wyżej okażą się nieprzydatne, a wtedy sieć zbuduje swoją regułę decyzyjną bez korzystania z tych „wątpliwej jakości" danych – czy też może lepiej nie „mieszać" do rozwiązania zadania zmiennych, które być może są bezwartościowe. Jaka jest Twoja opinia w tej sprawie? Odpowiedź uzasadnij.

12. **Zadanie dla zaawansowanych:** W literaturze dotyczącej sieci neuronowych czasami proponuje się, żeby struktura sieci (a więc zwłaszcza liczba warstw ukrytych i konfiguracja neuronów ukrytych) była ustalana z wykorzystaniem tak zwanego algorytmu genetycznego, naśladującego w komputerze proces biologicznej ewolucji, w wyniku której „przeżywają najlepiej przystosowani" (w tym przypadku „przeżyje" sieć mająca najlepsze właściwości w kontekście rozważanego zadania). Spróbuj dowiedzieć się więcej o tej metodzie, a potem ustal swój stosunek do niej: czy warto starać się „wyhodować drogą ewolucji" sieć o najkorzystniejszych właściwościach, czy lepiej do tej metody nie sięgać? Przedstaw argumenty przemawiające zarówno za jednym, jak i za drugim rozwiązaniem.

# 3. Uczenie sieci

## 3.1. Kim jest nauczyciel, który będzie uczyć sieć?

Cykl działania sieci neuronowej podzielić można na etap **nauki**, kiedy sieć gromadzi informacje potrzebne jej do określenia, co i jak ma robić, oraz na etap normalnego działania (nazywany czasem także egzaminem), kiedy w oparciu o zdobytą wiedzę sieć musi rozwiązywać konkretne **nowe** zadania (no bo rozwiązania zadań, których używałeś przy uczeniu sieci, specjalnie Cię nie interesują – rozwiązania tych zadań po prostu **znasz**). Kluczem do zrozumienia działania sieci i jej możliwości jest właśnie proces uczenia, zatem poznawanie sieci zaczniemy od omówienia tego właśnie procesu, natomiast działanie już nauczonych sieci różnych typów zostanie przedyskutowane w następnym rozdziale.

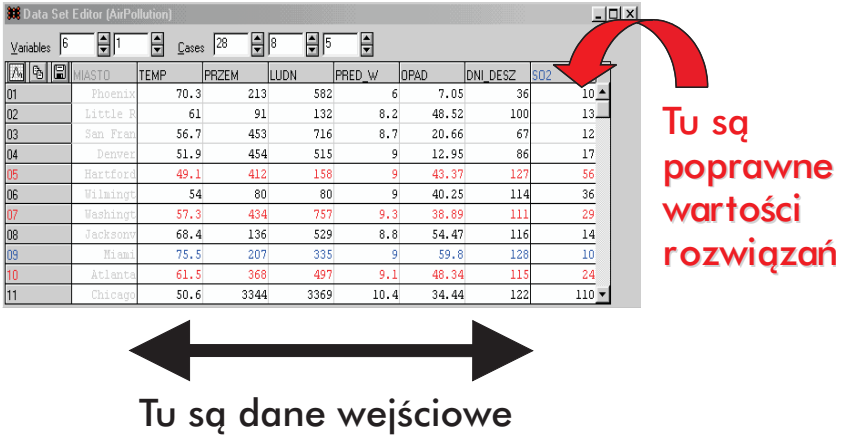
Możliwe są dwa warianty uczenia: **z nauczycielem** i **bez nauczyciela.** O uczeniu bez nauczyciela będziemy mówić oddzielnie w następnym podrozdziale, więc chwilowo zajmijmy się tylko modelem uczenia z nauczycielem. Uczenie takie polega na tym, że sieci podaje się przykłady poprawnego działania, które powinna ona potem naśladować w swoim bieżącym działaniu (w czasie egzaminu). **Przykład** należy rozumieć w ten sposób, że nauczyciel podaje konkretne sygnały wejściowe i wyjściowe, pokazując, jaka jest wymagana odpowiedź sieci dla pewnej konfiguracji danych wejściowych. Sieć obserwuje to, jaki jest związek między podanymi danymi wejściowymi a wymaganym wnioskiem (wynikiem, który powinna wyprodukować na wyjściu) i uczy się naśladować tę zasadę działania.

Pamiętaj: podczas uczenia z nauczycielem zawsze masz do czynienia z **parą** wartości – przykładowym sygnałem wejściowym i pożądanym (oczekiwanym) wyjściem, czyli wymaganą odpowiedzią sieci na ten sygnał wejściowy. Oczywiście, sieć może mieć wiele wejść i wiele wyjść, wtedy wspomnia-

na **para** oznacza w istocie dwa **zbiory** wartości: zbiór wartości, stanowiący kompletny zestaw danych wejściowych oraz zbiór wartości, które powinny się pojawić na wyjściu sieci jako pełne rozwiązanie zadania. Ale zawsze występują razem: **dane** dla zadania i jego **rozwiązanie**.

Wyjaśnijmy teraz, kim jest „nauczyciel" w metodzie uczenia „z nauczycielem". Otóż nazwa ta niekoniecznie oznacza człowieka, który własnoręcznie trenuje sieć – chociaż w następnych rozdziałach będziesz się tak bawił, że będziesz własnoręcznie trenował swoje sieci. W praktyce znacznie częściej rolę nauczyciela przejmuje jednak ten sam komputer, który modeluje rozważaną sieć neuronową, bo tak jest łatwiej i wygodniej. Niestety, sieci neuronowe nie są zbyt bystre i dla dobrego nauczenia sieci w jakimś trudniejszym zadaniu potrzeba kilkuset, a czasem nawet setek tysięcy pokazów! Kto miałby siłę i cierpliwość, żeby szkolić takiego nieuka!? Dlatego mówiąc o ***nauczycielu*** mam na myśli zawsze program komputerowy wyposażony przez człowieka w tak zwany **zbiór uczący.**

Czym jest zbiór uczący? Popatrz na rysunek 3.1.




Rys. 3.1. Przykładowy zbiór uczący

Widzisz na nim tabelę zawierającą pewne przykładowe dane. Są to dane dotyczące stopnia zanieczyszczenia powietrza w różnych miastach amerykańskich, ale równie dobrze mogły to być dowolne inne dane. Ważne jest, żeby to były dane rzeczywiste, wzięte z prawdziwej bazy danych. Zaznaczyłem to pozostawiając na rysunku 3.1 elementy oryginalnego okna pochodzącego z programu, który obsługiwał tę bazę danych. Wśród danych zebranych w bazie można wyodrębnić te, które będą wykorzystane jako wejścia do sieci (popatrz na zakres kolumn tabeli, wskazywany przez strzałkę na dole rysunku

3.1). W przykładzie prezentowanym na tym rysunku będą to dane pozwalające (być może) na przewidywanie stopnia zanieczyszczenia powietrza, a więc dane dotyczące ludności miasta, jego uprzemysłowienia, warunków atmosferycznych itd. Na podstawie tych danych, opisanych jako dane wejściowe, sieć będzie musiała przewidzieć, jaki będzie przeciętny stopień zanieczyszczenia powietrza w każdym mieście.

Dla miast, dla których ten stopień zanieczyszczenia nie był jeszcze badany, trzeba to będzie odgadnąć – i właśnie do tego celu użyjemy sieci, gdy już ją wspólnie nauczymy. Jednak na etapie uczenia sieci skorzystasz z faktu, że w bazie danych, wykorzystywanej jako zbiór uczący, dla pewnej liczby miast dane o zanieczyszczeniu powietrza są **znane** – i zostały umieszczone w odpowiedniej kolumnie tabeli, wskazanej na rysunku 3.1 czerwoną strzałką.

Masz więc na rysunku 3.1 dokładnie to, czego potrzebujesz, żeby nauczyć sieć: zbiór danych, w których podane są odpowiednie dane wejściowe i wyjściowe tworzące **pary**: widoczne są przyczyny (zaludnienie, uprzemysłowienie, warunki atmosferyczne) oraz skutek (wielkość zanieczyszczenia powietrza). Sieć skorzysta z tych danych i nauczy się prawidłowego działania (odgadywania zanieczyszczenia powietrza w miastach, w których nie dokonano jeszcze stosownych pomiarów), stosując jedną z wielu znanych dziś strategii uczenia. Przykładowe strategie uczenia będą niżej omówione nieco dokładniej. Natomiast, zanim to nastąpi, zwróć jeszcze uwagę na jeden szczegół rysunku 3.1: Otóż jedna kolumna tabeli jest tam pokazana z literami wyświetlonymi w kolorze szarym, a nie porządnie czarnym, tak że widać ją – ale słabo. Taki sposób zabarwienia sugeruje, że jest ona jak gdyby mniej ważna. I tak jest rzeczywiście: w tej kolumnie są dane dotyczące nazw poszczególnych miast. Informacje te są w bazie danych bardzo potrzebne, bo według nich wprowadza się do bazy nowe dane i wydobywa wyniki, ale dla sieci neuronowej taka informacja jest bezużyteczna (zanieczyszczenie powietrza nie zależy od tego, jaką **nazwę** ma rozważane miasto), więc chociaż odpowiednie dane są dostępne w bazie, to jednak nie będziemy ich używali do uczenia sieci. Takich nadmiarowych danych w bazie danych (pomijanych potem przy uczeniu sieci) może być więcej.

Zapamiętaj zatem, że „nauczycielem" przy uczeniu sieci będzie zazwyczaj jakiś zbiór danych, który jednak nie jest brany tak „jak leci", ale jest przystosowany do pełnienia roli zbioru uczącego poprzez odpowiednią konfigurację danych (musisz wyraźnie wskazać, których z nich będziesz używał jako dane wejściowe, a których jako dane wyjściowe) oraz poprzez ich roztropną selekcję: Nie należy „zaśmiecać" sieci danymi, które wprawdzie posiadasz, ale o których wiesz (lub podejrzewasz), że nie są użyteczne z punktu widzenia znajdowania rozwiązań postawionego zadania!

## 3.2. Czy sieć może uczyć się całkiem sama?

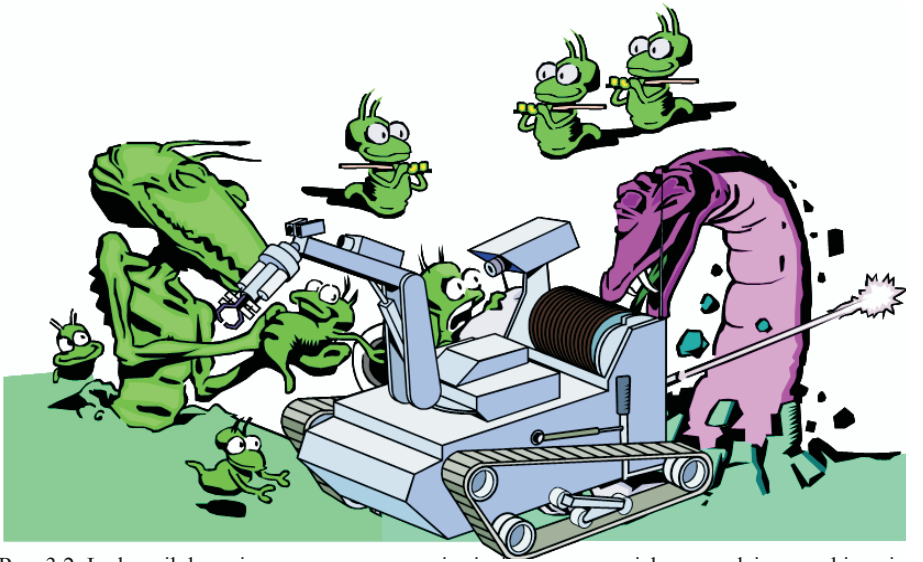
Obok opisanego wyżej schematu uczenia z nauczycielem jest w użyciu także szereg metod tak zwanego uczenia bez nauczyciela (albo **samouczenia** sieci). Metody te polegają na podawaniu na wejście sieci wyłącznie szeregu przykładowych danych wejściowych, bez podawania jakiejkolwiek informacji dotyczącej pożądanych czy chociażby tylko oczekiwanych sygnałów wyjściowych. Okazuje się, że odpowiednio zaprojektowana sieć neuronowa potrafi wykorzystać same tylko obserwacje wejściowych sygnałów i zbudować na ich podstawie sensowny algorytm swojego działania – najczęściej polegający na tym, że automatycznie wykrywane są klasy powtarzających się (być może z pewnymi odmianami) sygnałów wejściowych i sieć uczy się (zupełnie spontanicznie, bez jawnego nauczania) **rozpoznawać** te typowe wzorce sygnałów.

Przy samouczeniu sieci także musisz mieć zbiór uczący – tyle tylko że w zbiorze tym będą wyłącznie dane przewidziane do wprowadzenia na **wejście** sieci. Nie ma danych wyjściowych, bo technikę uczenia bez nauczyciela stosuje się w sytuacji, kiedy **nie wiesz, czego wymagać od sieci analizującej jakieś dane.** Gdyby jako przykład wziąć dane pokazane na rysunku 3.1, to do uczenia bez nauczyciela wykorzystałbyś wyłącznie kolumny opisane jako dane wejściowe, natomiast nie podawałbyś do sieci informacji z kolumny zaznaczonej czerwoną strzałką. Sieć, która zdobywałaby wiedzę w takim procesie uczenia, nie miałaby szans na przewidywanie, w którym mieście będzie większe, a w którym mniejsze zanieczyszczenie powietrza, bo takiej wiedzy sieć sama nie zdobędzie. Jednak analizując dane na temat różnych miast sieć może na przykład wyróżnić (całkiem sama!) grupę dużych miast przemysłowych i nauczyć się odróżniać te miasta od małych miasteczek będących centrami regionów rolniczych. To rozróżnienie da się wydedukować z danych wejściowych na tej zasadzie, że miasta przemysłowe są do siebie wzajemnie podobne, a miasta rolnicze mają z kolei także wiele wspólnych cech. Na podobnej zasadzie sieć może całkiem sama wyodrębnić miasta o dobrej i o złej pogodzie oraz dokonać wielu jeszcze innych klasyfikacji, opierając się wyłącznie na wartościach obserwowanych danych wejściowych.

Zauważ, że samouczenie sieci jest bardzo interesujące z punktu widzenia analogii, jakie istnieją między tego rodzaju działaniem sieci a działaniem ludzkiego umysłu: człowiek też ma zdolność spontanicznego klasyfikowania napotykanych obiektów i zjawisk (niektórzy nazywają to „formowaniem pojęć"), a po dokonaniu stosownej klasyfikacji rozpoznaje kolejne obiekty jako należące do jednej z tych wcześniej poznanych klas. Samouczenie jest też bardzo interesujące z punktu widzenia zastosowań, gdyż nie wymaga żadnej

jawnie podawanej do sieci neuronowej zewnętrznej wiedzy – która może być niedostępna lub której zgromadzenie może być zbyt fatygujące – a sieć zgromadzi wszystkie potrzebne informacje i wiadomości **całkiem sama**. W jednym z dalszych rozdziałów bardzo dokładnie opiszę Ci (i pokażę poglądowo poprzez odpowiednie programy!), na czym polega to samouczenie sieci.

Teraz możesz sobie wyobrazić (raczej dla zabawy i pobudzenia wyobraźni, niż z realnej potrzeby), że sieć samoucząca się z kamerą telewizyjną może być wysłana w bezzałogowej sondzie kosmicznej na Marsa. Nie wiemy, jakie tam panują warunki, nie wiemy więc, jakie obiekty nasza sonda powinna rozpoznawać. Mało tego – nie wiemy nawet, ile klas obiektów się pojawi! Nie szkodzi, sieć sobie poradzi (patrz rys. 3.2). Sonda ląduje i sieć zaczyna proces samouczenia. Na początku nie rozpoznaje nic, tylko obserwuje. Z czasem jednak proces spontanicznej samoorganizacji doprowadzi do tego, że sieć nauczy się wykrywać i rozróżniać między sobą różne typy sygnałów wejściowych, które pojawiają się na jej wejściu: oddzielnie skały i kamienie, oddzielnie formy roślinne (jeśli będą), a oddzielnie żywe organizmy. Jeśli damy sieci wystarczająco dużo czasu – może się tak wyszkolić, że potrafi odróżniać Marsjan od Marsjanek – chociaż jej twórca nie wiedział nawet, że oni istnieją!



Rys. 3.2. Lądownik kosmiczny z samouczącą się siecią neuronową jako narzędzie pozyskiwania wiedzy na temat obiektów, których istnienia konstruktor sieci nie mógł z góry przewidzieć

Oczywiście, opisany wyżej samouczący się lądownik marsjański jest wyłącznie tworem hipotetycznym, chociaż sieci same tworzące i rozpoznające różne wzorce istnieją i są chętnie stosowane. Na przykład, może nas interesо-

wać, ile i jakie formy pewnej mało znanej choroby w istocie występują. Czy jest to jedna jednostka chorobowa, czy kilka? Czym się one różnią? Jak je leczyć?

Wystarczy wtedy wziąć samouczącą się sieć neuronową i dostatecznie długo pokazywać jej informacje o rejestrowanych pacjentach i ich symptomach. Po pewnym czasie sieć poda informację, ile typowych grup symptomów i objawów da się wykryć i na podstawie jakich kryteriów można odróżniać pacjentów zaliczanych do różnych grup. Do badacza należy wtedy tylko nadanie tym grupom odpowiednio mądrze brzmiących łacińskich nazw chorób – i już może biec do krawca, żeby przymierzyć frak uszyty na uroczystość wręczenia Nagrody Nobla!

Opisana metoda samouczenia ma oczywiście (jak wszystko na tym świecie) określone wady – opiszę je jednak nieco później, kiedy już będzie jasne, jak to wszystko działa. Ma ona jednak tak wiele niezaprzeczalnych zalet, że dziwić się należy jej stosunkowo małej popularności!

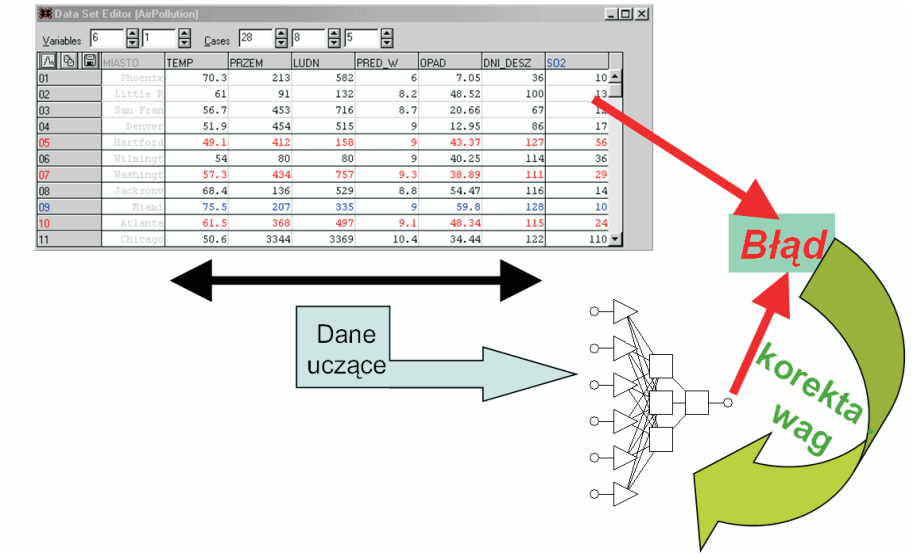
## 3.3. Gdzie i jak sieci neuronowe gromadzą zdobytą wiedzę?

Przeanalizujmy teraz bliżej proces uczenia z nauczycielem. Jak to się dzieje, że sieć zdobywa i gromadzi wiedzę? Otóż kluczowym pojęciem są tu opisane w poprzednim rozdziale **wagi** wejść poszczególnych neuronów. Przypomnijmy: każdy neuron ma wiele wejść, za pomocą których odbiera sygnały od innych neuronów oraz sygnały wejściowe podawane do sieci jako dane do obliczeń. Z wejściami tymi skojarzone są parametry nazywane wagami; każdy sygnał wejściowy jest najpierw przemnażany przez wagę, a dopiero później sumowany z innymi sygnałami. Jeśli zmienią się wartości wag – neuron zacznie pełnić innego rodzaju funkcję w sieci, a co za tym idzie – cała sieć zacznie inaczej działać. **Cała sztuka uczenia sieci polega więc na tym, by tak dobrać wagi, żeby wszystkie neurony wykonywały dokładnie takie czynności, jakich się od nich wymaga.**

Neuronów w sieci mogą być tysiące, każdy ma setki wejść – niepodobna więc dla wszystkich tych wejść zdefiniować potrzebne wagi w sposób jednorazowy i arbitralny własnoręcznie. Można jednak zaprojektować i zrealizować **proces uczenia** polegający na rozpoczęciu działania sieci z pewnym przypadkowym zestawem wag i na stopniowym **polepszaniu** tych wag. W każdym kroku procesu uczenia wartości wag jednego lub kilku neuronów ulegają zmianie, przy czym reguły tych zmian są tak pomyślane, by każdy neuron **sam** potrafił określić, które ze swoich wag ma zmienić, w którą stronę (zwiększenie lub zmniejszenie), a także o ile. Oczywiście, przy określaniu potrzebnych zmian wag neuron może korzystać z informacji pochodzących od

nauczyciela (o ile stosujemy uczenie z nauczycielem), nie zmienia to jednak faktu, że sam proces zmiany wag (będących w sieci jedynym śladem pamięciowym) przebiega w każdym neuronie sieci w sposób **spontaniczny i niezależny,** dzięki czemu może być realizowany bez konieczności bezpośredniego stałego dozoru ze strony osoby sterującej tym procesem. Co więcej, proces uczenia jednego neuronu jest niezależny od tego, jak uczy się dowolny inny neuron, więc uczenie może być prowadzone **równocześnie** we wszystkich neuronach sieci (oczywiście pod warunkiem zbudowania odpowiedniej sieci jako układu elektronicznego, a nie w formie programu symulacyjnego). Pozwala to uzyskiwać bardzo duże szybkości uczenia i zaskakująco dynamiczny wzrost „kwalifikacji" sieci, która dosłownie mądrzeje na naszych oczach!

Podkreślę jeszcze raz, bo ma to kluczowe znaczenie: nauczyciel nie musi wnikać w szczegóły procesu uczenia – wystarczy, że poda sieci wzór poprawnego rozwiązania. Sieć porówna swoje rozwiązanie, jakie sama uzyskała dla




Rys. 3.3. Typowy krok uczenia sieci neuronowej

wykorzystywanego przykładu pochodzącego z danych uczących, z tym rozwiązaniem, które jest zapisane w zbiorze uczącym jako wzorcowe (a więc zapewne[1] poprawne). Algorytmy uczenia są tak zbudowane, że wiedza o war-

[1] Nauczyciel uczący sieć nie musi być idealny, to znaczy może się czasem mylić, podając dla niektórych danych niepoprawne rozwiązania – a sieć i tak się nauczy poprawnie znajdować odpowiedzi na stawiane jej pytania. Tę zdumiewającą zdolność sieci neuronowych, które potrafią często po nauczeniu mylić się **rzadziej** niż zawodny nauczyciel, który je nauczał, jako pierwszy wykrył i opisał Frank Rosenblatt w zbudowanym przez siebie „Perceptronie". Ma to duże znaczenie praktyczne, bowiem przy kompletowaniu zbioru uczącego często zdarza się,

tości **błędu**, jaki sieć popełnia, wystarcza do tego, żeby skorygować wartości jej wag, przy czym każdy neuron z osobna (sterowany wspomnianym algorytmem) koryguje swoje wagi na wszystkich wejściach całkiem sam – jeśli tylko dostanie wiadomość, jaki błąd popełnił. Jest to bardzo prosty a jednocześnie skuteczny mechanizm, pokazany symbolicznie na rysunku 3.3. Jego systematyczne stosowanie powoduje, że sieć krok po kroku doskonali swoje działanie, aż wreszcie potrafi rozwiązać wszystkie zadania ze zbioru uczącego, a na podstawie uogólnienia tej wiedzy – potrafi także rozwiązywać inne zadania, które zostaną jej przedstawione na etapie „egzaminu".

Opisany wyżej sposób uczenia sieci jest najczęściej stosowany, ale w niektórych zadaniach (na przykład przy rozpoznawaniu obrazów) nie musi się podawać sieci dokładnej **wartości** pożądanego sygnału wyjściowego, ale do skutecznego uczenia wystarczą same tylko ogólne informacje na temat tego, czy aktualne jej zachowanie jest poprawne, czy nie. Czasami mówi się wręcz o sygnałach „nagrody" i „kary", na podstawie których wszystkie neurony sieci same znajdują i wprowadzają właściwe poprawki do swego działania. Ta analogia do tresury zwierząt nie jest wcale przypadkowa!

## 3.4. Jak zorganizować naukę sieci?

Potrzebne zmiany wielkości współczynników wagowych w poszczególnych neuronach obliczane są w oparciu o specjalne reguły (nazywane czasem *paradygmatami* sieci), przy czym liczba różnych używanych dziś reguł i ich odmian jest prawdziwie astronomiczna, gdyż prawie każdy badacz usiłował wnieść do dziedziny sieci neuronowych swój wkład w formie nowej reguły uczenia. Pewien pogląd na ten temat daje zbiór algorytmów uczenia opisanych w mojej książce *Sieci neuronowe*, do której stale odsyłam bardziej dociekliwych Czytelników po bardziej szczegółowe informacje (http://winntbg.bg.agh.edu.pl/skrypty/0001/ ). Tutaj omówię krótko (bez stosowania matematyki, taka była umowa zawarta zaraz na początku książki!) dwie podstawowe reguły uczenia: regułę najszybszego spadku, leżącą u podstaw większości algorytmów uczenia z nauczycielem oraz regułę Hebba, stanowiącą najprostszy przykład uczenia bez nauczyciela.

**Reguła najszybszego spadku** polega na tym, że każdy neuron otrzymawszy na swoich wejściach określone sygnały (z wejść sieci albo od innych neuronów, stanowiących wcześniejsze piętra przetwarzania informacji) wyznacza

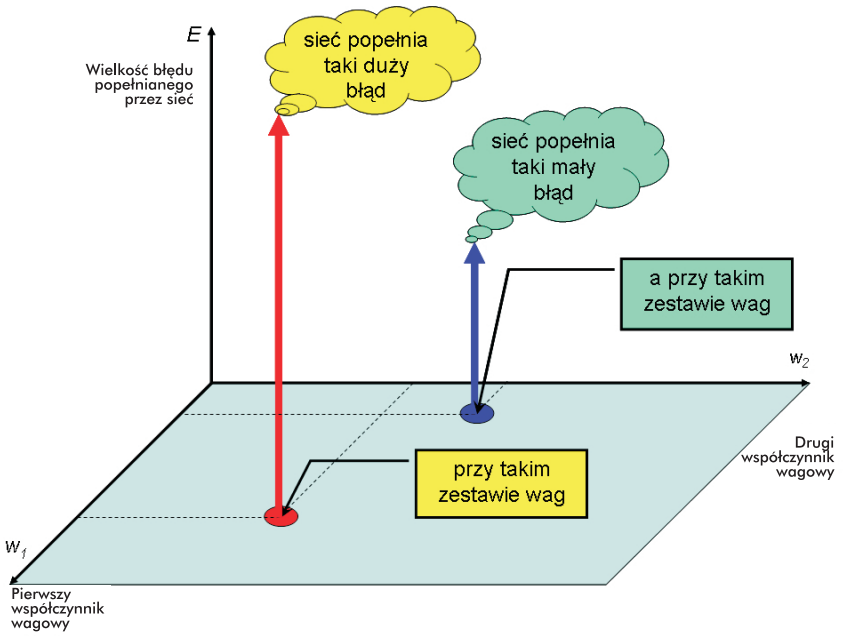
---

że niektóre przykłady (pochodzące na przykład z badań przeprowadzanych na jakimś rzeczywistym obiekcie) nie są całkiem pewne. Wspomniana **odporność** sieci na błędy nauczyciela pozwala na skuteczne ich uczenie nawet w takim przypadku, gdy **niektóre** (oczywiście niezbyt liczne!) przykłady ze zbioru uczącego zawierają błędy!

swój sygnał wyjściowy wykorzystując posiadaną wiedzę w postaci wcześniej ustalonych wartości współczynników wzmocnienia (wag) wszystkich wejść oraz (ewentualnie) progu. Sposoby wyznaczania przez neurony wartości sygnałów wyjściowych na podstawie sygnałów wejściowych omówione zostały dokładniej w poprzednim rozdziale. Wartość sygnału wyjściowego, wyznaczonego przez neuron w danym kroku procesu uczenia, porównywana jest z odpowiedzią wzorcową podaną przez nauczyciela w ciągu uczącym. Jeśli występuje rozbieżność (a na początku procesu uczenia prawie zawsze taka rozbieżność się pojawi, bo skąd niby ten neuron ma wiedzieć, czego my od niego chcemy?) – neuron wyznacza różnicę pomiędzy swoim sygnałem wyjściowym a tą wartością sygnału, która byłaby – według nauczyciela – prawidłowa, a także ustala (za pomocą metody najszybszego spadku, którą zaraz wyjaśnię), jak trzeba zmienić wartości wag, żeby ten błąd najszybciej zmalał.

W dalszych rozważaniach użyteczne będzie pojęcie **powierzchni błędu**, które teraz wprowadzimy i dokładnie omówimy. Otóż wiesz już o tym, że działanie sieci zależy od wartości współczynników wagowych neuronów będących jej elementami. Jeśli znasz zestaw wszystkich współczynników wagowych, występujących we wszystkich neuronach całej sieci, to wiesz, jak się taka sieć zachowa. W szczególności możesz takiej sieci pokazać (kolejno) wszystkie dostępne Ci przykłady zadań wraz z rozwiązaniami, wchodzące w skład zbioru uczącego. Za każdym razem, gdy sieć wyznaczy swoją odpowiedź na zadane jej pytanie – możesz porównać ją z wzorcem prawidłowej odpowiedzi, który znajduje się w zbiorze uczącym, wyznaczając błąd, jaki sieć popełniła. Miarą tego błędu jest zwykle różnica pomiędzy wartością wyniku dostarczoną przez sieć oraz wartością wyniku odczytanego ze zbioru uczącego. Żeby ocenić całościowo działanie sieci przy określonym zestawie współczynników wagowych w jej neuronach – stosuje się zwykle wzór na sumę **kwadratów** błędów popełnianych przez sieć dla poszczególnych przypadków ze zbioru uczącego. Błędy przed sumowaniem podnosi się do kwadratu po to, żeby uniknąć efektu wzajemnego kompensowania się błędów dodatnich i ujemnych, a ponadto podnoszenie do kwadratu powoduje, że szczególnie wysoka „kara" spotyka sieć za **duże** błędy (dwa razy większy błąd to cztery razy większy składnik w tworzonej sumie).

Popatrz na rysunek 3.4. Pokazana została na nim sytuacja, jaka by mogła mieć miejsce w tak małej sieci neuronowej, że miałaby ona tylko dwa współczynniki wagowe. Tak małych sieci neuronowych nie ma, ale **wyobraź sobie,** że masz właśnie taką sieć, bo tylko dla takiej maleńkiej sieci uda się narysować jej zachowanie bez wchodzenia w trudne, wielowymiarowe przestrzenie. Każdy stan lepszego albo gorszego wyuczenia tej sieci wiązać się będzie z jakimś punktem poziomej (jasnoniebieskiej) płaszczyzny widocznej

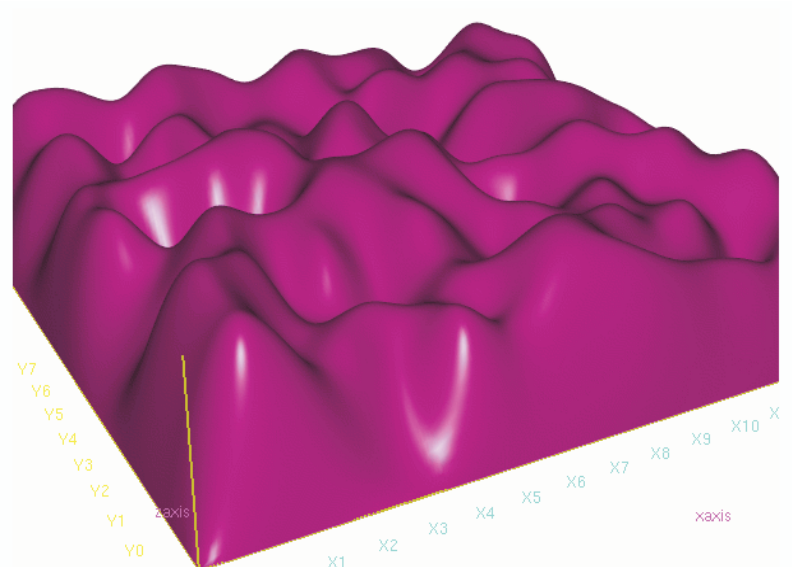



Rys. 3.4. Sposób konstruowania powierzchni błędu (przy czytaniu proszę zacząć od objaśnień na żółtym tle)

na rysunku wraz z jej współrzędnymi, to znaczy obydwoma rozważanymi współczynnikami wagowymi. Wyobraź sobie teraz, że ustawiłeś w sieci takie wartości współczynników wagowych, jakie odpowiadają lokalizacji czerwonego punktu na płaszczyźnie. Egzaminując taką sieć za pomocą wszystkich elementów zbioru uczącego ustalisz sumaryczną wartość błędu tej sieci – i w miejscu, w którym był czerwony punkt wystawisz do góry strzałkę (czerwoną!) o wysokości odpowiadającej obliczonej wartości błędu (zgodnie z opisem pionowej osi na rysunku).

Następnie wybierz inne wartości wag, wyznaczające inne położenie punktu na płaszczyźnie (granatowy) – i wykonaj te same czynności, otrzymując granatową strzałkę.

A teraz wyobraź sobie, że te czynności wykonujesz dla **wszystkich** kombinacji współczynników wagowych, czyli dla wszystkich punktów jasnoniebieskiej płaszczyzny. W jednych miejscach błędy będą większe, w innych mniejsze, co mógłbyś zobaczyć (gdybyś miał cierpliwość tak wiele razy egzaminować swoją sieć) w postaci pofałdowanej **powierzchni błędu,** rozpościerającej się ponad płaszczyzną zmienianych wag. Przykład takiej powierzchni pokazałem Ci na rysunku 3.5.




Rys. 3.5. Przykładowa powierzchnia błędu przy uczeniu sieci neuronowej

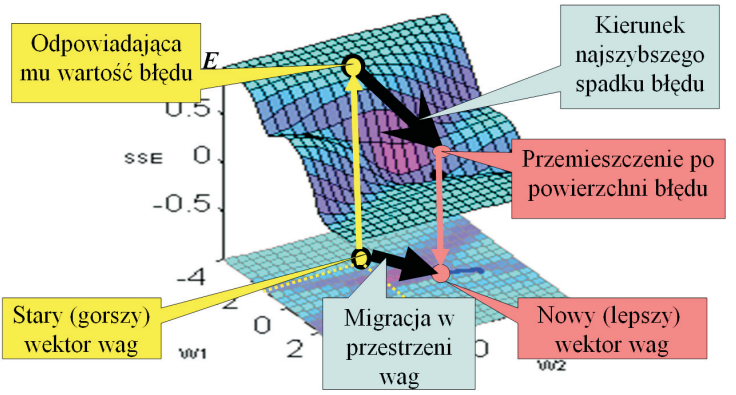
Jak widać na tej powierzchni jest wiele „pagórków" – są to miejsca, w których sieć popełnia szczególnie dużo błędów, w związku z czym takich miejsc trzeba unikać, oraz jest wiele głębokich dolin, które nas bardzo interesują, bo tam właśnie, na dnie takich dolin, sieć neuronowa popełnia mało błędów, czyli dobrze rozwiązuje postawione jej zadanie.

Jednak jak znaleźć taką dolinę?

Otóż uczenie sieci neuronowej powinieneś rozpatrywać jako wieloetapowy **proces.** Podczas tego procesu krok po kroku będziesz starał się **poprawiać** wartości wag w sieci, zmieniając stare (gorsze) zestawy wag, powodujące, że sieć popełnia duży błąd, na nowe, co do których będziesz miał nadzieję (ale nie pewność!), że są lepsze. Popatrz na rysunek 3.6, na którym starałem się to zilustrować.

Zaczynasz od sytuacji pokazanej w lewym dolnym narożniku rysunku, to znaczy masz jakiś stary zestaw (wektor) wag, zaznaczony na płaszczyźnie parametrów sieci żółtym kółkiem. Dla tego wektora wag wyznaczasz błąd, jaki sieć popełnia, i „lądujesz" na powierzchni błędu w miejscu, które wskazuje żółta strzałka w lewym górnym rogu rysunku. O tej sytuacji nie da się nic dobrego powiedzieć: błąd jest bardzo duży, więc sieć ma chwilowo bardzo złe parametry. Trzeba to poprawić.

Jak?




Rys. 3.6. Istota procesu uczenia polega na posuwaniu się w kierunku spadku powierzchni błędu

Ano właśnie. Metody uczenia sieci neuronowych potrafią ustalić, w którą stronę trzeba zmienić współczynniki wagowe, żeby uzyskać efekt zmniejszenia błędu. Taki kierunek najszybszego spadku błędu zaznaczony jest na rysunku 3.6 dużą czarną strzałką. Niestety, szczegółów tego, jak metody uczenia to robią, nie da się objaśnić bez używania skomplikowanej matematyki i takich pojęć jak *gradient* albo *pochodna cząstkowa,* jednak wnioski z tych dość skomplikowanych rozważań matematycznych są dosyć proste. Otóż każdy neuron w sieci dokonuje modyfikacji swoich współczynników wagowych (i ewentualnie progu), stosując następujące dwie proste reguły:

⇒ wagi zmieniane są tym silniej, im większy błąd został wykryty;

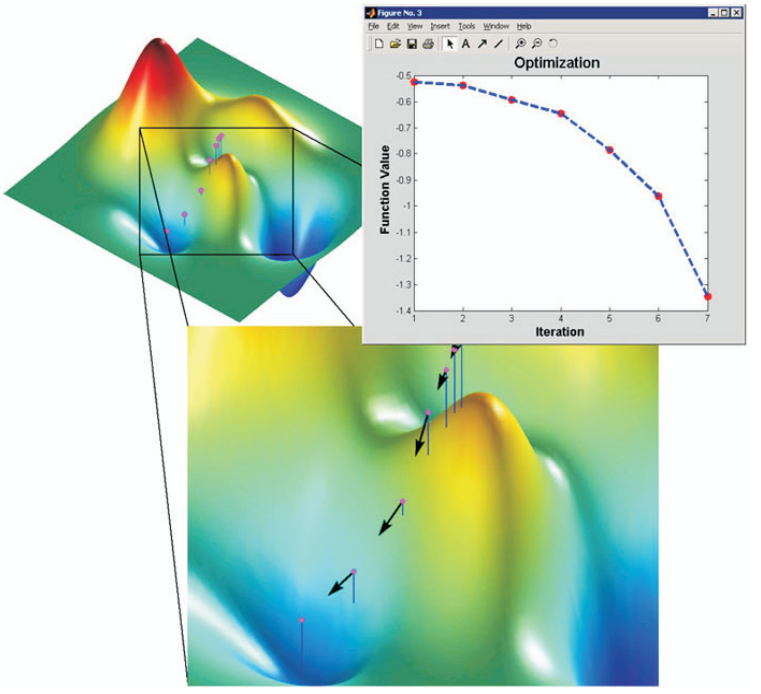
⇒ wagi związane z tymi wejściami, na których występowały duże wartości sygnałów wejściowych, zmieniane są bardziej niż wagi tych wejść, na których sygnał wejściowy był niewielki.

Opisane wyżej podstawowe reguły wymagają jeszcze w praktyce kilku dodatkowych korekt (za chwilę powiem o nich więcej), jednak sam opisany zarys metody uczenia jest chyba jasny. Znając błąd popełniony przez neuron oraz znając jego sygnały wejściowe, możesz łatwo przewidzieć, jak będą się zmieniać jego wagi. Zauważ także, jak bardzo logiczne i sensowne są te matematycznie wyprowadzone reguły: Na przykład, bezbłędna reakcja neuronu na podany sygnał wejściowy powinna oczywiście powodować zachowanie jego wag bez zmian – wszak doprowadziły one do sukcesu. I tak właśnie się dzieje!

Zauważ, że sieć stosując opisane metody w praktyce sama przerywa proces uczenia gdy jest już dobrze wytrenowana, gdyż małe błędy powodują jedynie minimalne, „kosmetyczne“ korekty wag. Jest to logiczne, podobnie jak zasada uzależniania wielkości korekty od wielkości wejściowego sygnału

przekazywanego przez rozważaną wagę – wszak te wejścia, na których występowały większe sygnały, miały większy wpływ na wynik działania neuronu, który okazał się błędny, trzeba je więc silniej „temperować". W szczególności opisany algorytm powoduje, że dla wejść, na których w danym momencie nie były podawane sygnały (podczas obliczeń miały one zerowe wartości), odpowiednie wagi nie są zmieniane, nie wiadomo bowiem, czy są dobre, czy nie, gdyż nie uczestniczyły w tworzeniu aktualnego (błędnego, skoro trzeba coś poprawiać) sygnału wyjściowego.

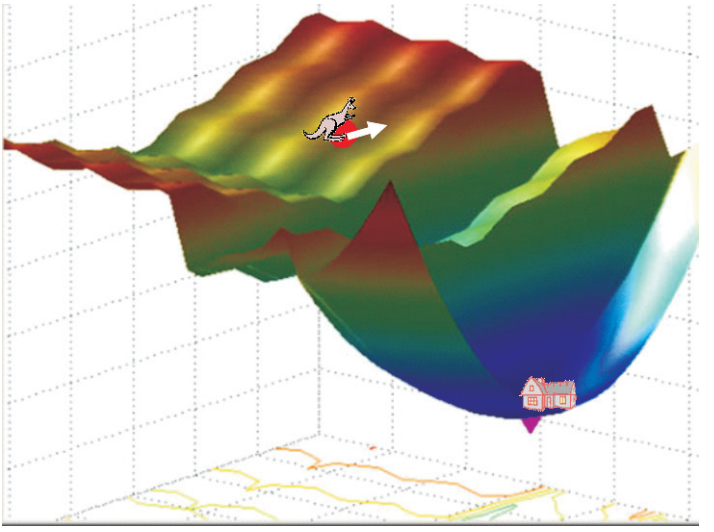
Wracając do prezentacji jednego kroku procesu uczenia, pokazanej na rysunku 3.6, zauważ, co się dalej dzieje: Znalazłszy kierunek najszybszego spadku błędu algorytm uczenia sieci dokonuje migracji w przestrzeni wag, polegającej na zmianie starego (gorszego) wektora wag na nowy (lepszy). Migracja ta powoduje, że na powierzchni błędu „ześlizgniemy się" do nowego punktu – najczęściej położonego niżej, czyli przybliżającego sieć do upragnionej doliny, w której błędy są najmniejsze, a rozwiązanie postawionego zadania – najdoskonalsze. Taki optymistyczny scenariusz stopniowego i skutecznego podążania w kierunku miejsca, gdzie błędy są najmniejsze, pokazuje rysunek 3.7.




Rys. 3.7. Poszukiwanie (i znajdowanie!) minimum funkcji błędu metodą wyznaczania w każdym kolejnym kroku kierunku najszybszego spadku

## 3.5. Dlaczego to się czasem nie udaje?

Nie zawsze uczenie jest takie proste i przyjemne, jak to pokazałem na rysunku 3.7. Czasami sieć się długo „szamocze" zanim dojdzie do wymaganego rozwiązania. Dokładne omówienie pojawiających się tu trudności znowu wymagałoby odwołania się do skomplikowanych rozważań na temat rachunku różniczkowego i różnicowego wraz z analizą zbieżności algorytmów, i innymi raczej trudnymi rzeczami. Zamiast jednak prześladować Cię skomplikowaną matematyką, spróbuję Ci wyjaśnić, o co właściwie chodzi, opowiadając Ci historię o niewidomym kangurze (patrz rysunek 3.8).



Rys. 3.8. Uczenie sieci neuronowej jako wędrówka ślepego kangura szukającego domu w najgłębszej dolinie. Kangur wie, w którą stronę opada teren, ale nie wie, jak daleko może bezpiecznie skoczyć

Wyobraź sobie, że niewidomy kangur zgubił się w górach i pragnie powrócić do swego domu, o którym wie tylko tyle, że mieści się na samym dnie najgłębszej doliny. Kangur wymyśla prostą metodę: Jest wprawdzie ślepy, więc nie może zobaczyć całego pejzażu otaczających go gór (podobnie, jak algorytm uczenia sieci nie może sprawdzić wartości funkcji błędu we wszystkich punktach dla dowolnych zestawów wartości wagowych), ale może wymacać łapkami, w którą stronę teren opada (tak samo jak algorytm uczenia sieci potrafi ustalić, w którą stronę trzeba zmienić wagi, żeby błąd zmalał). No więc kangur znajduje właściwy kierunek i hop! – skacze, ile ma sił w nogach, licząc na to, że zmierza do swego domku.

Niestety – na kangura (i na algorytm uczenia sieci neuronowej) czekają różne niespodzianki. Jak obejrzysz dokładnie rysunek 3.8, to zauważysz, że nieostrożny skok może zaprowadzić kangura na dno rozpadliny, która oddziela go od domku. Możliwa jest także sytuacja (nie narysowana, ale łatwa do wyobrażenia), że teren może w pewnym kierunku opadać – ale nieco dalej gwałtownie się będzie wznosił, co spowoduje, że wykonując daleki skok w pozornie obiecującym kierunku – kangur w istocie pogorszy swoją sytuację, bo znajdzie się wyżej (czyli dalej od celu), niż był w momencie startu!

Sukces biednego kangurka zależy głównie od tego, czy potrafi dobrze wymierzyć długość skoku. Jak będzie wykonywał małe skoki, to droga do domu zajmie mu bardzo dużo czasu. Ale jak się zdecyduje na za długi skok, a w otoczeniu są jakieś urwiska albo rozpadliny, to zrobi sobie krzywdę!

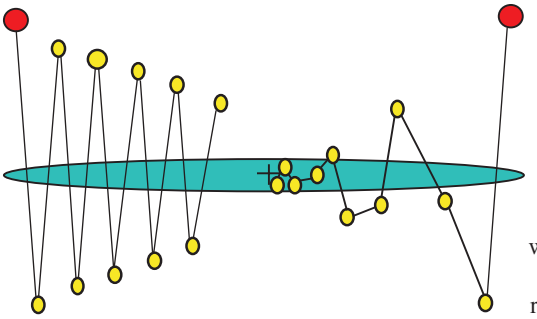
Przy uczeniu sieci twórca algorytmu musi także zdecydować, jak silne powinny być zmiany wag powodowane przez określone wartości sygnałów wejściowych i określoną wielkość błędu. Decyduje o tym współczynnik proporcjonalności zwany *learning rate*. Na pozór może on być wybierany dowolnie, jednak każda konkretna decyzja ma określone konsekwencje. Wybranie współczynnika **za małego** prowadzi do bardzo powolnego procesu uczenia (wagi są poprawiane w każdym kroku bardzo słabo, żeby więc osiągnęły pożądane wartości, trzeba wykonać bardzo dużo takich kroków). Z kolei wybór **za dużego** współczynnika uczenia powoduje bardzo gwałtowne zmiany parametrów sieci, które w krańcowym przypadku prowadzić mogą nawet do niestabilności procesu uczenia (sieć miota się, nie mogąc znaleźć prawidłowych wartości wag, które zmieniają się za szybko, by precyzyjnie „wstrzelić się" w potrzebne rozwiązanie).

Można na to popatrzeć także z jeszcze jednego punktu widzenia. Duże wartości współczynnika uczenia odpowiadają postawie bardzo wymagającego i surowego nauczyciela, który zbyt radykalnie i surowo karze ucznia za popełniane błędy. Taki nauczyciel rzadko osiąga dobre wyniki nauczania, bo wprawia uczniów w zmieszanie i wywołuje nadmierny stres. Z kolei małe wartości tego współczynnika uczenia odpowiadają charakterystyce nauczyciela nadmiernie wyrozumiałego, którego uczniowie robią zbyt wolne postępy, bo ich niewystarczająco pogania do roboty.

Przy uczeniu sieci i przy uczeniu uczniów konieczny jest więc wybór kompromisowy, uwzględniający zarówno korzyści związane z szybką pracą, jak i względy bezpieczeństwa, wskazujące na konieczność uzyskania stabilnej pracy procesu uczenia. Można jednak pomóc sobie w jeszcze jeden sprytny sposób, który opiszę Ci w kolejnym podrozdziale.

## 3.6. Do czego służy momentum?

Pewnym sposobem zwiększenia szybkości uczenia bez naruszania stabilności jest zastosowanie do algorytmu uczenia dodatkowego składnika, tzw. *momentum*. Obrazowo można powiedzieć, że momentum zwiększa bezwładność procesu uczenia – zmiany wag zależą wtedy zarówno od błędów popełnianych przez sieć w danym momencie, jak i od przebiegu procesu uczenia we wcześniejszym okresie.



Rys. 3.9. Zmiany współczynników wag w sieci podczas uczenia bez współczynnika momentum (po lewej) i ze współczynnikiem momentum równym 0.5 (po prawej) dla tej samej ilości kroków

Rysunek 3.9 pozwala porównać proces uczenia z momentum i bez. Na rysunku tym pokazano przebieg zmian współczynników wag (trzech – rysunek należy interpretować jako prezentację w perspektywie procesu, jaki ma miejsce w trójwymiarowej przestrzeni wag) dla pewnego neuronu sieci. Czerwone punkty przedstawiają punkty startowe (wynikające z początkowych – przed uczeniem – wartości współczynników wag), a żółte punkty to wartości współczynników wag uzyskiwane w kolejnych krokach procesu uczenia. Założono, że minimum funkcji błędu osiągane jest w punkcie „+”, a niebieska elipsa pokazuje kontur stałego błędu (zbiór wartości współczynników wag, dla których proces uczenia osiąga ten sam poziom błędu). Jak widać na rysunku, wprowadzenie momentum rzeczywiście powoduje, że proces uczenia staje się spokojniejszy (wartości współczynników wag nie zmieniają się tak gwałtownie i tak często), a ponadto efektywniejszy (kolejne punkty szybciej zbliżają się do punktu „+” będącego rozwiązaniem problemu). Obecnie podczas uczenia sieci **z reguły** korzysta się z momentum, ponieważ usprawnia to proces dochodzenia do poprawnych rozwiązań, a jednocześnie nie jest zbyt kosztowne.

Inny sposób polepszenia procesu uczenia polegać może na stosowaniu zmiennych wartości współczynników uczenia – małych na początku procesu uczenia, gdy sieć wybiera dopiero kierunki swego działania, większych

w środkowej części uczenia, kiedy trzeba dość forsownie ale jeszcze zgrubnie dostosować wartości parametrów sieci do założonych zasad jej działania, i wreszcie ponownie mniejszych pod koniec procesu uczenia, w momencie kiedy sieć dopracowuje ostateczne wartości swoich parametrów (*fine tuning*) i zbyt gwałtowne korekty mogą zburzyć konstrukcję wcześniej zbudowanej struktury wiedzy. Zauważmy, że te techniki działania, o matematycznie wyprowadzonej strukturze i empirycznie przebadanej przydatności, żywo przypominają metody wypracowane przez nauczycieli o dużej praktyce dydaktycznej i stosowane do uczniów o małej odporności psychicznej! Jest to bez wątpienia zastanawiająca zbieżność zachowania sieci neuronowej i ludzkiego umysłu – nie pierwsza, i nie ostatnia zresztą.

## 3.7. Od czego zacząć uczenie sieci?

Problemem, z którym musi sobie poradzić każdy twórca algorytmu uczenia sieci neuronowej, jest kwestia początkowych (startowych) wartości współczynników wag. Opisany wyżej algorytm uczenia pokazuje, jak można **poprawiać** wartości tych współczynników w trakcie procesu uczenia, od czegoś jednak ten proces trzeba zacząć, gdyż już w momencie pokazania pierwszego obiektu ciągu uczącego sieć musi mieć jakieś konkretne, ustalone wartości współczynników wag, by móc określić wartości sygnałów wyjściowych wszystkich neuronów, a następnie porównać je z wartościami zadanymi przez nauczyciela w celu określenia błędów. Skąd jednak wziąć te początkowe wartości wszystkich wag?

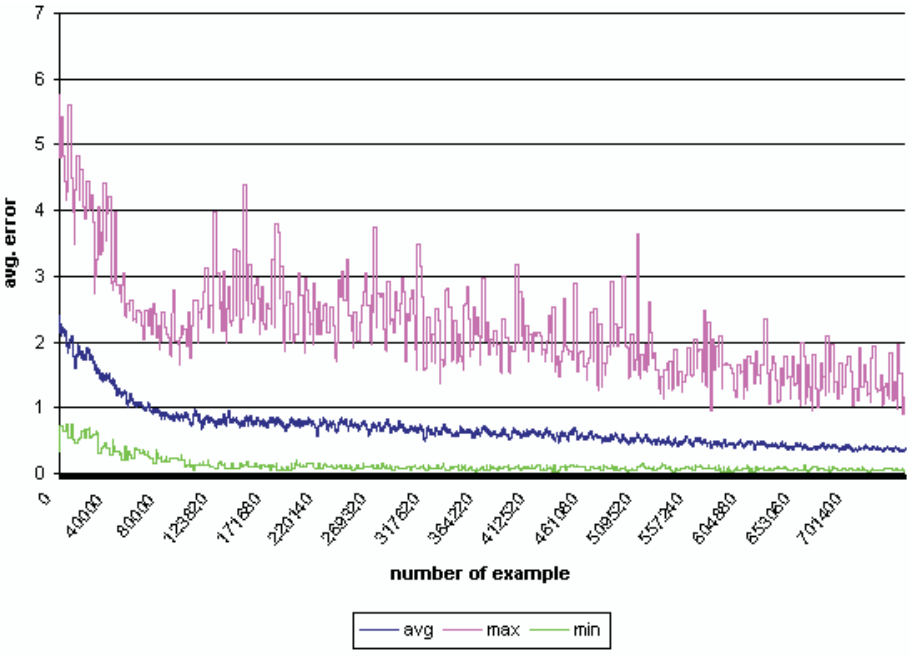
Otóż teoria uczenia sieci neuronowych mówi, że przy spełnieniu pewnych (dość prostych) warunków sieć liniowa potrafi nauczyć się i znaleźć prawidłowe wartości ostatecznych współczynników wagowych (rozwiązujących postawiony problem, jeśli tylko takie rozwiązanie istnieje) **niezależnie** do tego, od jakich wartości początkowych wystartujemy proces uczenia. W dziedzinie sieci nieliniowych sytuacja nie jest tak prosta, gdyż proces uczenia może w nich utknąć w tzw. minimach lokalnych funkcji błędu, co w praktyce oznacza, że startując od różnych punktów początkowych (od różnych początkowych zestawów współczynników wag) możemy otrzymywać **różne** parametry nauczonej sieci – lepiej lub gorzej nadające się do rozwiązywania postawionego zadania. Tak mówi teoria, nie mówi ona natomiast nic na temat tego, jak wybrać te dobre, korzystne punkty startowe?

Tu musimy niestety polegać na empirii. Prosta i chętnie używana praktyka polega na tym, że początkowe wartości współczynników wagowych przyjmujemy w sposób **losowy.** Na początku wszystkie ustawiane parametry otrzymują więc wartości przypadkowe, to znaczy startujemy z jakiegoś nie dającego

się z góry przewidzieć miejsca w przestrzeni wag. To się może w pierwszej chwili wydawać dziwne, ale to ma sens – jeśli nie wiadomo, gdzie szukać potrzebnego minimum funkcji błędu, to nie ma lepszego rozwiązania jak tylko zdać się na ślepy los!

W programach symulacyjnych jest do tego celu specjalne polecenie „zasiewające" w sieci początkowe losowe wartości współczynników, przy czym unikać należy ich zbyt dużych wartości (rutynowo stosowany jest przedział od –0,1 do +0,1, z którego są one wybierane w sposób losowy). Dodatkowo w sieciach wielowarstwowych unikać także trzeba współczynników mających wartość 0, gdyż blokuje to proces nauki głębiej położonych warstw (przez połączenia z zerową wartością wagi sygnały nie przechodzą, co oznacza w praktyce „odcięcie" części struktury sieci od dalszej nauki).

Postępowi uczenia sieci towarzyszy oczywiście zmniejszanie się błędu. Proces ten z początku jest bardzo szybki, potem jednak ulega spowolnieniu, aż wreszcie po wykonaniu pewnej liczby kroków – proces uczenia zatrzymuje się zupełnie. Warto pamiętać, że proces uczenia mający na celu wytrenowanie wykonywania tego samego zadania za pomocą tej samej sieci – może przebiegać różnie. Na rysunku 3.10. pokazano trzy przebiegi tego procesu, które róż-




Rys. 3.10. Zmniejszanie się błędu w trakcie procesu uczenia dla różnych przypadków uczenia tej samej sieci

nią się zarówno szybkością postępu uczenia, jak i końcowym efektem (w postaci wielkości współczynnika błędu, poniżej którego sieć już nie chce zejść, mimo intensywności dalszego uczenia). Ponieważ wszystkie wykresy dotyczą tej samej sieci i tego samego zadania, a różnią się tylko początkowymi (przypadkowo wybieranymi przed uczeniem) wartościami współczynników wag – można powiedzieć, że na wykresach tych ujawniają się zróżnicowane „wrodzone zdolności" sieci.

W niektórych typach sieci (na przykład *Kohonena*) wymaga się dodatkowo, by początkowe wartości współczynników wagowych były w pewien sposób unormowane, ale o tym porozmawiamy podczas omawiania tych właśnie szczególnych typów sieci.

## 3.8. Czy sieć trzeba długo uczyć?

Niestety, odpowiedź na postawione wyżej pytanie jest pesymistyczna. Za wygodę korzystania z procesu uczenia zamiast „ręcznego" programowania sieci trzeba zapłacić długim czasem uczenia – i na ogół trzeba ten fakt przyjąć z pokorą, bo nie znaleziono na to żadnej naprawdę skutecznej rady. Co gorsza, trudno z góry przewidzieć, jak długo trzeba będzie uczyć **daną konkretną sieć**, zanim zacznie ona przejawiać jakieś elementy inteligentnego zachowania. Czasem idzie to błyskawicznie, ale na ogół długo i mozolnie trzeba pokazywać elementy ciągu uczącego, zanim sieć zacznie chociaż trochę rozumieć, czego od niej wymagamy. Co więcej, na skutek losowego wyboru początkowych wartości współczynników wagowych wyniki uczenia **tej samej sieci** dla tych samych danych użytych jako ciąg uczący mogą się nieco od siebie różnić i z tym faktem trzeba się także po prostu pogodzić – chyba że ktoś ma czas na to, by tę samą sieć uczyć kilkakrotnie i potem wybierać ten wariant, który gwarantuje najlepszy efekt. W praktyce jest to jednak bardzo czasochłonne, gdyż w badaniach moich i moich doktorantów stwierdziłem wielokrotnie, że jednorazowe nauczenie sieci o kilku tysiącach elementów trwa **kilkadziesiąt godzin!**

Nie jest to wynik odosobniony. Na przykład z zestawienia publikowanego w miesięczniku *Electronic Design* wynika, że liczba prezentacji ciągu uczącego, koniecznych do uzyskania poprawnego działania sieci w wybranych zastosowaniach, wynosi:

- dla rozpoznawania znaków Kenji – $10^{13}$
- dla rozpoznawania mowy – $10^{12}$
- dla rozpoznawania rękopisów – $10^{12}$
- dla syntezy mowy – $10^{10}$
- dla prognozowania finansowego – $10^{9}$

Inni autorzy przytaczają podobne wyniki.

Ze względu na szybkość uczenia poszukuje się więc odmiennych (od klasycznego opisywanego tu algorytmu delta) nowych metod uczenia sieci. Uważa się, że czas uczenia sieci algorytmem najszybszego spadku (opisanym wyżej) rośnie wykładniczo ze wzrostem liczby elementów sieci, co może stanowić w przyszłości jeden z zasadniczych czynników limitujących rozmiary budowanych sieci.

## 3.9. Jak uczyć warstwy ukryte?

Na koniec przedyskutujemy jeszcze problem uczenia sieci związany z obecnością w niej warstw ukrytych. Jak opisałem w poprzednim rozdziale – warstwą ukrytą zwykło się nazywać każdą warstwę z wyjątkiem wejściowej i wyjściowej. Istota problemu pojawiającego się przy uczeniu neuronów warstwy ukrytej polega na tym, że dla neuronów tych nie da się bezpośrednio wyznaczyć wielkości błędów, ponieważ nauczyciel podaje wzorcowe wartości sygnałów jedynie dla warstwy wyjściowej, natomiast sygnałów wyznaczonych przez neurony warstwy ukrytej (też zapewne na początku uczenia błędnych) nie ma z czym porównać. Pewnym rozwiązaniem dla tego problemu jest metoda tak zwanej wstecznej propagacji błędu (*backpropagation*), zaproponowana po raz pierwszy przez *Paula Werbosa,* a potem spopularyzowana przez *Davida Rumelharta* i wiązana zwykle z jego imieniem.

Metoda ta polega na odtwarzaniu przypuszczalnych wartości błędów głębszych (ukrytych) warstw sieci na podstawie rzutowania wstecz błędów wykrytych w warstwie wyjściowej. Z grubsza można powiedzieć (w tej chwili – bo w dalszych rozdziałach książki proces ten poddany będzie bardzo szczegółowej analizie), że robi się to w ten sposób: rozważając każdy kolejny neuron warstwy ukrytej bierze się pod uwagę błędy wszystkich tych neuronów, do których wysyłał on swój sygnał wyjściowy i **sumuje się je,** uwzględniając wielkości współczynników wag połączeń między rozważanym neuronem a neuronami, których błędy są sumowane. W ten sposób „odpowiedzialność" za błędy neuronów warstwy ukrytej obciąża neuron warstwy ukrytej tym silniej, im silniej (bo z większą wagą) wpływał sygnał danego neuronu ukrytego na wypracowanie określonych sygnałów warstwy wyjściowej (czyli na powstanie określonych błędów wykrywanych przy ocenie odpowiedzi sieci).

Postępując konsekwentnie w ten sposób i posuwając się krok po kroku od wyjścia do wejścia sieci można wyznaczyć przypuszczalne błędy wszystkich neuronów, a tym samym uzyskać przesłanki pozwalające określić potrzebne poprawki współczynników wagowych tych neuronów. Oczywiście, opisane postępowanie nie pozwala nigdy wyznaczyć błędów neuronów warstw pośrednich w sposób idealnie dokładny, zatem korekty błędów w trakcie procesu

uczenia są mniej trafne i mniej dokładne niż w przypadku sieci bez warstw ukrytych. Przejawia się to w praktyce znacznie wydłużonym czasem uczenia takich sieci, o czym była wyżej mowa. Biorąc jednak pod uwagę znacznie szerszy zakres możliwości sieci wielowarstwowych, uznać trzeba, że warto ponieść koszt zwiększonej złożoności procesu uczenia, nawet jeśli potrzebny wynik (w postaci poprawnie wytrenowanej sieci) uda się osiągnąć dopiero po kilkudziesięciu tysiącach pokazów wszystkich obiektów ciągu uczącego.

Z wielokrotnym prezentowaniem w trakcie uczenia sieci neuronowej wszystkich obiektów ciągu uczącego wiążą się pewne dodatkowe problemy. Konieczność takiego prezentowania ciągu uczącego „w kółko" wynika z faktu, że nawet przy założeniu świętej cierpliwości nauczyciela można zgromadzić jako materiał do uczenia sieci kilkadziesiąt, czasem kilkaset, a w sporadycznych wypadkach kilka tysięcy przykładów – nigdy jednak nie setki tysięcy czy miliony, a tylu właśnie kroków uczenia wymagają często trenowane sieci. Nie ma więc wyjścia – trzeba te same obiekty (to znaczy sygnały wejściowe i wzorce odpowiedzi sieci) pokazywać wiele razy. Zdecydowanie niekorzystne jest jednak pokazywanie ciągu uczącego stale w tej samej kolejności jego elementów. Takie periodyczne nauczanie może prowadzić do pojawiania się określonych cykli wartości współczynników wagowych, co nie posuwa naprzód procesu uczenia i było przyczyną wielu znanych mi osobiście niepowodzeń. Bardzo ważna jest więc randomizacja procesu uczenia – przypadkowe mieszanie ciągu uczącego i pokazywanie sieci jego elementów za każdym razem w innej kolejności. Komplikuje to nieco proces uczenia, ale w większości zadań jest warunkiem uzyskania sensownych wyników.

## 3.10. W jaki sposób sieć może się uczyć sama?

Postawione w tytule tego podrozdziału pytanie kieruje nas do zagadnień znanych w literaturze pod nieco przewrotną nazwą „uczenia bez nauczyciela". Koncepcja zadania, które ma być rozwiązywane, sprowadza się do tego, by sieć sama z siebie doskonaliła sposób rozwiązywania określonego zadania nie mając do dyspozycji gotowych wzorów zadań wraz z rozwiązaniami. Czy to jest jednak w ogóle możliwe? Jak można oczekiwać, że sieć będzie doskonaliła swoje działanie, nie dając jej w tym zakresie żadnych wskazówek?

Otóż okazuje się, że jest w pełni możliwe i nawet stosunkowo łatwo osiągalne spontaniczne samoorganizowanie się sieci neuronowej – przy spełnieniu kilku prostych warunków. Najprostsza koncepcja samouczenia sieci oparta jest na spostrzeżeniu fizjologa i psychologa amerykańskiego, **Donalda Hebba**, że w mózgu zwierząt (i w umyśle człowieka) zachodzą procesy wzmacniania połączeń między neuronami (lub całymi ich zespołami, tak

zwanymi ośrodkami), pod warunkiem że zostały one pobudzone do pracy **równocześnie**. W ten sposób powstają skojarzenia, tak (zdaniem Hebba) kształtują się odruchy, tak również formują się proste formy umiejętności ruchowych i percepcyjnych.

Przenosząc teorie Hebba na grunt sieci neuronowych, informatycy zaprojektowali metodę samouczenia polegającą na tym, że sieci pokazuje się kolejne przykłady sygnałów wejściowych, **nie podając żadnych informacji o tym, co z tymi sygnałami należy zrobić.** Sieć po prostu obserwuje otoczenie i odbiera różne sygnały, nikt jej jednak nie mówi, jakie znaczenie mają pokazujące się obiekty i jakie są pomiędzy nimi zależności. Sieć na podstawie obserwacji występujących sygnałów stopniowo sama odkrywa, jakie jest ich znaczenie i również sama ustala zachodzące między sygnałami zależności. W ten sposób sieć nie tylko **nabywa** wiedzę – ona ją poniekąd sama **tworzy!** Prześledźmy jednak teraz ten ciekawy proces nieco dokładniej.

Po podaniu do sieci neuronowej każdego kolejnego zestawu sygnałów wejściowych tworzy się w tej sieci pewien **rozkład** sygnałów wyjściowych – niektóre neurony sieci są pobudzone bardzo silnie, inne słabiej, a jeszcze inne mają sygnały wyjściowe wręcz ujemne. Interpretacja tych zachowań może być taka, że niektóre neurony „rozpoznają" podawane sygnały jako „własne" (czyli takie, które są skłonne akceptować), inne traktują je „obojętnie", zaś jeszcze u innych neuronów wzbudzają one wręcz „awersję". Po ustaleniu się sygnałów wyjściowych wszystkich neuronów w całej sieci – wszystkie wagi wszystkich neuronów są zmieniane, przy czym wielkość odpowiedniej zmiany wyznaczana jest na podstawie **iloczynu sygnału wejściowego,** wchodzącego na dane wejście (to, którego wagę zmieniamy) **i sygnału wyjściowego** produkowanego przez neuron, w którym modyfikujemy wagi. Łatwo zauważyć, że jest to właśnie realizacja postulatu Hebba: w efekcie opisanego wyżej algorytmu połączenia między źródłami silnych sygnałów i neuronami, które na nie silnie reagują – są wzmacniane.

Dokładniejsza analiza procesu samouczenia metodą Hebba pozwala stwierdzić, że w wyniku konsekwentnego stosowania opisanego algorytmu początkowe, najczęściej przypadkowe „preferencje" neuronów ulegają systematycznemu wzmacnianiu i dokładnej polaryzacji. Jeśli jakiś neuron miał „wrodzoną skłonność" do akceptowania sygnałów pewnego rodzaju – to w miarę kolejnych pokazów nauczy się te sygnały rozpoznawać coraz dokładniej i coraz bardziej precyzyjnie. Po dłuższym czasie takiego samouczenia w sieci powstaną zatem (całkiem spontanicznie!) **wzorce** poszczególnych **typów** występujących na wejściu sieci sygnałów. W wyniku tego procesu sygnały podobne do siebie będą w miarę postępu uczenia coraz skuteczniej **grupowane i rozpoznawane** przez pewne neurony, zaś inne typy sygnałów

staną się „obiektem zainteresowania" innych neuronów. W wyniku tego procesu samouczenia sieć – całkiem sama, tylko obserwując podawane do niej sygnały – nauczy się, ile klas podobnych do siebie sygnałów pojawia się na jej wejściach oraz sama przyporządkuje tym klasom sygnałów neurony, które nauczą się je rozróżniać, rozpoznawać i sygnalizować. **Tylko** tyle i **aż** tyle!

## 3.11. Jak prowadzić samouczenie?

Opisany wyżej proces samouczenia ma niestety swoje wady. W porównaniu z procesem **uczenia z nauczycielem** samouczenie jest zwykle znacznie powolniejsze, zatem jeśli ma się możliwość wyboru – należy raczej wybierać uczenie sterowane, a nie spontaniczne. Co więcej, bez nauczyciela nigdy nie wiemy z góry, **który** neuron wyspecjalizuje się w rozpoznawaniu której klasy sygnałów (na przykład pierwszy może uparcie rozpoznawać literę R, drugi D, trzeci G – i żadna siła nie zmusi tych neuronów do ustawienia się w porządku alfabetycznym). Stanowi to pewną trudność przy wykorzystywaniu i interpretacji wyników pracy sieci – na przykład w systemie sterowania robota. Wskazana trudność ma zresztą jeszcze bardziej podstawowy charakter i bardziej dokuczliwe skutki – nie ma mianowicie gwarancji, że rozwijając swoje początkowe, losowe preferencje, neurony wyspecjalizują się tak dalece, że każdy z nich wskazywać będzie **inną** klasę wejściowych obrazów. Przeciwnie, jest wysoce prawdopodobne, że kilka neuronów uprze się rozpoznawać tę samą klasę sygnałów – na przykład kilka neuronów rozpoznawać będzie literę A, żaden natomiast nie „zdecyduje się" rozpoznawać litery B. Dlatego sieć przeznaczona do samouczenia musi być **większa** niż sieć wykonująca to samo zadanie, ale trenowana w sposób klasyczny, z udziałem nauczyciela. Trudno to dokładnie wymierzyć, ale z moich własnych doświadczeń i z obserwacji studentów wykonujących prace magisterskie w moim laboratorium zdaje się wynikać, że sieć przeznaczona do samouczenia musi mieć przynajmniej **trzykrotnie** więcej elementów (zwłaszcza warstwy wyjściowej), niż by to wynikało z liczby odpowiedzi, jakich sieć po nauczeniu powinna udzielać.

Bardzo subtelną i istotną kwestią jest wybór początkowych wartości wag neuronów sieci przeznaczonej do samouczenia. Wartości te mają bardzo silny wpływ na ostateczne zachowanie sieci, ponieważ proces uczenia jedynie pogłębia i doskonali pewne tendencje istniejące w sieci od samego początku, przeto od jakości tych początkowych, „wrodzonych" właściwości sieci silnie zależy, do czego sieć dojdzie na końcu procesu uczenia. Nie wiedząc z góry, jakiego zadania sieć powinna się uczyć, trudno wprowadzać jakikolwiek zdeterminowany mechanizm nadawania początkowych wartości wag, jednak pozostawienie wszystkiego wyłącznie mechanizmom losowym może

powodować, że sieć (zwłaszcza mała) może nie zdołać wystarczająco zróżnicować swego działania w początkowym okresie procesu uczenia i wszelkie późniejsze wysiłki, by znaleźć w strukturze sieci reprezentację dla wszystkich występujących w wejściowych sygnałach klas, mogą okazać się daremne. Można jednak wprowadzić pewien mechanizm wstępnego „rozprowadzania" wartości wag w początkowej fazie procesu uczenia. Metoda ta, zwana *convex combination* (opisana dokładniej w mojej książce *Sieci neuronowe* dostępnej w Internecie pod adresem ***http://winntbg.bg.agh.edu.pl/skrypty/0001/*** ) modyfikuje początkowe wartości wag w taki sposób, by zwiększyć prawdopodobieństwo równomiernego pokrycia przez poszczególne neurony wszystkich typowych sytuacji pojawiających się w wejściowym zbiorze danych. Jeśli tylko dane pojawiające się w początkowej fazie uczenia nie będą różniły się istotnie od tych, jakie sieć będzie potem analizować i różnicować – to metoda *convex combination* stworzy w sposób automatyczny dogodny punkt wyjścia do dalszego samouczenia i zapewni stosunkowo dobrą jakość nauczonej sieci w większości praktycznych zadań.

## 3.12. Pytania kontrolne i zagadnienia do samodzielnego rozwiązania

1. Jakie warunki musi spełniać zbiór danych, żeby mógł być użyty jako zbiór uczący dla sieci neuronowej?
2. Czym różni się uczenie sieci „z nauczycielem" od uczenia „bez nauczyciela"?
3. Na jakiej podstawie algorytm uczący wyznacza kierunek i wielkość zmiany wag w kolejnych krokach procesu uczenia?
4. W jaki sposób należy nadać początkowe wartości współczynnikom wagowym sieci, żeby zapoczątkować proces jej uczenia?
5. Kiedy proces uczenia sam się zatrzyma?
6. Jakie są powody niepowodzeń w procesie uczenia sieci i jak im można zapobiegać?
7. Na czym polega proces uogólniania wiedzy zdobytej przez sieć neuronową w trakcie procesu uczenia?
8. Skąd można uzyskać dane niezbędne do ustalania w trakcie uczenia wielkości zmian współczynników wagowych neuronów wchodzących w skład **warstw ukrytych**?
9. Do czego służy *momentum* i co można osiągnąć z jego pomocą?
10. Jak długiego uczenia wymagają duże sieci neuronowe? Od czego to zależy?

11. Dlaczego w przypadku sieci samouczących się mówi się często o „odkrywaniu wiedzy", a nie po prostu o nauce sieci?

12. **Zadanie trudniejsze, dla zaawansowanych:** Spróbuj zaproponować regułę, zgodnie z którą można by było zmieniać współczynnik uczenia (*learning rate*) w trakcie procesu uczenia w taki sposób, żeby przyspieszać uczenie w sytuacji, gdy jest to bezpieczne, i spowalniać w momencie, gdy zagrożona jest stabilność uczenia.

**Wskazówka:** Rozważ z m i a n y wartości błędów popełnianych przez sieć w kolejnych krokach procesu uczenia.

# 4. Działanie najprostszej sieci

## 4.1. Jak przejść od teorii do praktyki, czyli jak używać programów przeznaczonych dla czytelników tej książki?

Poprzednie trzy rozdziały służyły do tego, by dostarczyć Ci podstawowych wiadomości teoretycznych na temat sieci neuronowych. Od tego rozdziału natomiast zaczniesz nabywać praktyki w ich używaniu. Od tej chwili bowiem będę starał się ilustrować wszystkie dalsze wywody za pomocą prostych programów, które pozwolą Ci własnoręcznie spróbować, jak działają sieci neuronowe i jakie są podstawowe reguły ich używania. Programy te (a także potrzebne do ich uruchomienia dane) znajdują się w Internecie na stronie:

**http://www.agh.edu.pl/tad**

Kiedy wpiszesz ten adres do swojej przeglądarki WWW, Twoim oczom ukaże się strona, która dokładnie, krok po kroku, opisuje wszystkie czynności, jakie będziesz musiał wykonać, żeby sobie wziąć (za darmo ale całkiem legalnie!) te programy, którymi dalej będziemy się wspólnie posługiwali w tej książce. Programy te będą potrzebne, żebyś nie musiał bezkrytycznie wierzyć w to, co Ci będę pisał o sieciach neuronowych. Mając te programy Ty sam będziesz mógł odkrywać na swoim komputerze różne właściwości sieci neuronowych, wykonując bardzo ciekawe doświadczenia i bawiąc się przy tym znakomicie. Bez obaw – wszystko, co będziesz musiał zrobić, to uruchomić jeden lub dwa typowe programy instalacyjne. Jeśli więc kiedykolwiek instalowałeś cokolwiek na swoim komputerze, na pewno będziesz to potrafił zrobić. Szczegółowa instrukcja instalacji znajduje się jednak właśnie na podanej stronie WWW, gdyż oprogramowanie bywa uaktualniane i zbyt szczegółowy opis w książce mógłby szybko stać się nieaktualny. Kilka nudnych (niestety!) szczegółów, dotyczących tego, co powinieneś zrobić, pozwolę sobie jednak przytoczyć.

Pobranie samych programów ze strony, której adres Ci wyżej podałem, jest bardzo łatwe i wykonuje się za pomocą jednego kliknięcia (w razie potrzeby poproś o pomoc kogoś, kto już kiedyś pobierał jakieś zasoby z Internetu). Jednak same programy nie wystarczą. Programy te napisane są w języku C# i wymagają zainstalowania bibliotek, które się nazywają .Net Framework (w wersji 2.0). Nie przejmuj się, jeśli chwilowo nie masz pojęcia, o co chodzi – całe wymagane oprogramowanie znajdziesz pod podanym adresem w Internecie.

Pierwszym krokiem, jaki powinieneś wykonać, jest wspomniana wyżej instalacja bibliotek. Oczywiście może się zdarzyć, że ten krok będzie niepotrzebny, bo w Twoim komputerze są już zainstalowane te biblioteki. Jeśli jesteś tego pewien, wtedy możesz pominąć krok instalacji .Net Framework, ale na wszelki wypadek radzę Ci go wykonać. Jeśli właściwe programy są już w komputerze, to instalator sam wykryje, że nie trzeba ich instalować. Może się jednak zdarzyć, że na stronie o podanym wyżej adresie internetowym znajdują się nowsze wersje bibliotek, niż te, które Ty masz u siebie. Wtedy instalator wykona jednak swoją pracę, co należy oceniać pozytywnie, bo zawsze warto skorzystać z okazji i wymienić starsze oprogramowanie na bardziej nowoczesne. To nowe oprogramowanie może Ci się przydać do wielu dalszych celów, nie tylko związanych z czytaniem tej książki i wykonywaniem opisanych w niej eksperymentów.

Bardzo ważna uwaga: Całe oprogramowanie udostępnione na stronie **http://www.agh.edu.pl/tad**
jest legalne i bezpłatne w ramach edukacyjnej licencji udzielonej przez firmę Microsoft®.
Licencja ta pozwala na bezpłatne używanie i rozwijanie technologii .NET jeśli nie jest ona używana do celów nie komercyjnych. Oznacza to, że możesz korzystać bez ograniczeń z tego oprogramowania w ramach doświadczeń opisanych w tej książce, a także możesz je używać do tworzenia Twoich własnych programów - przy założeniu, że nie będziesz tych programów potem sprzedawał.

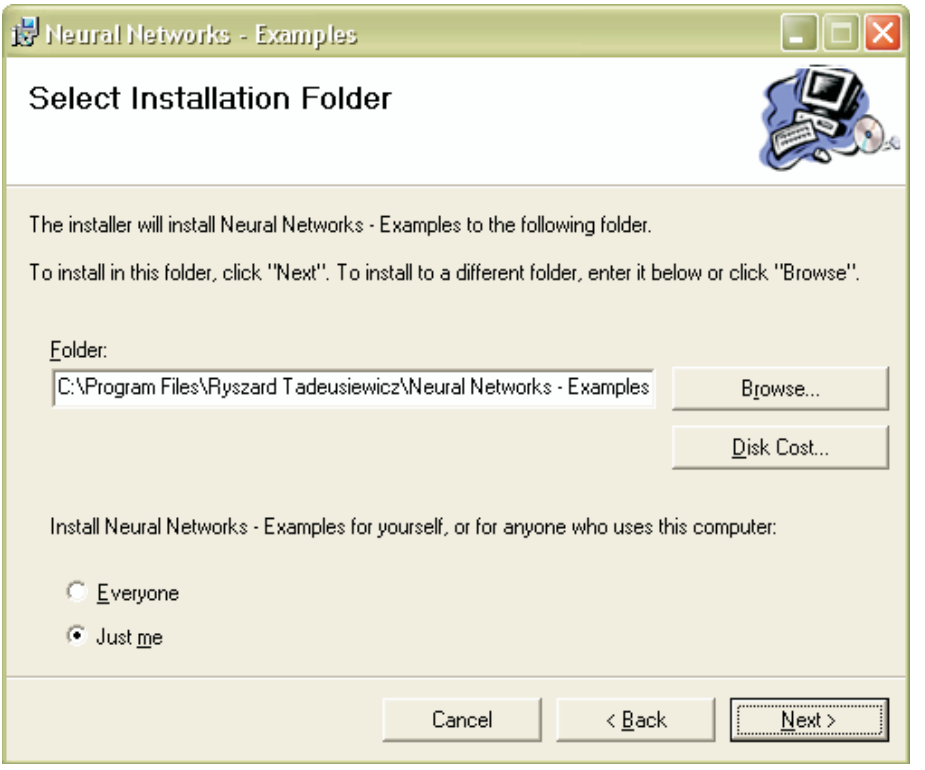
Powróćmy do procesu instalacji potrzebnego Ci oprogramowania.

Kiedy już skończy się instalacja bibliotek **.Net Framework**, jako niezbędnego składnika wszelkich dalszych Twoich działań, zainstaluj przykładowe programy według wskazówek umieszczonych na wskazanej wyżej stronie WWW. Ponownie uruchomi się instalator, który pozwoli Ci automatycznie (a więc z Twojego punktu widzenia – całkiem bezboleśnie…) zainstalować wszystkie napisane przeze mnie przykładowe programy. Z programów tych

będziesz korzystał podczas dalszego czytania książki, i dzięki nim będziesz mógł nie tylko teoretycznie poznać sieci neuronowe, ale dodatkowo praktycznie się nimi pobawić (poważniej mówiąc: przeprowadzić niezbędne eksperymenty ☺ ) i przy okazji nabrać bardzo korzystnej praktyki w stosowaniu mechanizmów obliczeń neuronowych. Kiedy instalator skończy pracę, będziesz miał do nich dostęp przez menu Start, tak jak do większości innych Twoich programów.

Aby przekonać niedowiarków, że to proste, na rysunku 4.1 pokazany jest jedyny „trudny" moment instalacji, w którym instalator pyta użytkownika, gdzie zainstalować programy i dla kogo mają one być dostępne. Przy czym i tak najmądrzej jest zostawić wartości domyślne i po prostu kliknąć **Next**.

Po wykonaniu kilku zagadkowych czynności, których nie pojmują nawet najstarsi Górale, w Twoim menu **Start**, w podmenu **Programy**, powinna pojawić się grupa **Neural Networks – Examples**, zawierająca wszystko, co będzie Ci potrzebne do szczęścia. No, może poza kawą i białą czekoladą.




Rys. 4.1. Instalator zada Ci tak naprawdę tylko dwa pytania

Dla co bardziej dociekliwych adeptów informatyki (a do tych właśnie się zaliczasz, nieprawdaż?) przeznaczone są pozostałe dwie opcje: **Installation of source codes (Instalacja kodów źródłowych)** oraz **Installation Visual Studio 2005 Express** (**Instalacja zintegrowanego środowiska oprogramowania narzędziowego Visual Studio .NET**).

Pierwsza opcja pozwala skopiować na dysk twardy Twojego komputera czytelne dla człowieka (to znaczy dla Ciebie!) teksty tych wszystkich przykładowych programów, których wykonywalnymi wersjami będziesz się posługiwał podczas czytania książki. Jest to bardzo interesująca możliwość, pozwalająca Ci sprawdzić, jak to wszystko jest zrobione, a także jak to się dzieje, że każdy z wykorzystywanych programów działa tak jak działa. Posiadanie kodów źródłowych programów umożliwi Tobie – jeśli się odważysz – wprowadzenie własnych zmian do tych programów i prowadzenie jeszcze ciekawszych eksperymentów.

Jednak chcę podkreślić, że samo tylko używanie programów opisywanych w książce nie wymaga wcale dostępu do ich kodów źródłowych. Ale jeśli je będziesz miał, to będziesz mógł się więcej nauczyć i ciekawiej skorzystać z zasobów, jakie dla Ciebie przygotowałem. Warto się o to pokusić, zwłaszcza że pozyskanie tych kodów źródłowych nie jest wcale pracochłonne: Wystarczy po kliknięciu przycisku **Installation of source codes** wskazać folder, do którego zostaną skopiowane kody źródłowe. Ze względu na Twoją wygodę, folder ten powinien być szybko dostępny; najlepiej więc wybrać do tego celu **Moje dokumenty** lub coś równie „bliskiego", żeby nie trzeba było długo szukać na dysku, ilekroć będziesz chciał zajrzeć, „jak to jest zrobione".

Ostatnia opcja – **Instalacja zintegrowanego środowiska programowania Visual Studio .Net** – kierowana jest przede wszystkim do tych, którzy będą chcieli modyfikować nasze programy, aby przeprowadzić swoje własne eksperymenty, a być może również na bazie ich składników – napisać swoje własne programy. Słowem – jest to opcja dla ambitnych. Jednak zalecam w miarę możliwości skorzystanie z tej opcji również tym, którzy kody źródłowe będą tylko przeglądać. Warto poświęcić tych kilkanaście dodatkowych minut, aby przeglądanie programów stało się o wiele łatwiejsze, bo **Visual Studio .Net** to bardzo wygodne zintegrowane środowisko wytwarzania oprogramowania (ang. *Integrated Development Environment – IDE*) do przeglądania i edycji kodu, dołączania zewnętrznych zestawów, łatwego diagnozowania aplikacji i szybkiego generowania ostatecznych, skompilowanych wersji oprogramowania.

Pamiętaj, że instalacja zarówno kodów źródłowych, jak i **Visual Studio .Net** jest jak najbardziej dobrowolna i możesz z niej skorzystać albo nie. Do uruchomienia przykładowych programów, za pomocą których będziesz włas-

nymi rękami mógł tworzyć i badać opisywane w książce sieci neuronowe, wystarczy zainstalować **.Net Framework** (krok pierwszy) i same przykładowe programy (krok drugi).

No dobrze, skopiowałeś i zainstalowałeś wszystko, co trzeba – co dalej?

Aby skorzystać teraz z któregokolwiek z przykładów, wystarczy wybrać odpowiednie polecenie z menu **Start/Programy/Neural Networks – Examples.** Po uruchomieniu odpowiedniego programu będziesz mógł tworzyć i analizować w Twoim komputerze każdą kolejno opisywaną w książce sieć neuronową. Początkowo będzie to sieć o zaprogramowanych przeze mnie rozmiarach i kształcie, ale jak się zagłębisz w kodzie źródłowym, to będziesz mógł absolutnie wszystko zmodyfikować i zmienić. Uruchomiony program sprawi, że sieć będzie żyła na Twoim komputerze, można będzie ją uczyć, poddawać testom, analizować – słowem badać. Myślę, że taki sposób **odkrywania** właściwości sieci neuronowych – poprzez ich budowanie i zaprzęganie do pracy – może okazać się bardziej inspirujący i pobudzający wyobraźnię niż jakikolwiek wykład czy teoretyczny wywód.

Działanie sieci zależy silnie od jej struktury i przeznaczenia, dlatego będziemy musieli rozważyć w kolejnych rozdziałach kilka konkretnych przypadków szczególnych. Ponieważ jednak najprościej można wyjaśnić działanie sieci służącej do **rozpoznawania obrazów** (a poza tym na rozpoznawaniu obrazów ja się dobrze znam, a na innych zastosowaniach sieci – niekoniecznie)**,** dlatego od niej właśnie zaczniemy dyskusję.

Sieć taka – jak już była mowa – na wejściu otrzymuje obraz, a na wyjściu ma zasygnalizować, do której z wcześniej wyuczonych klas należy obiekt widoczny na obrazie. Sieć taką przedstawiał na przykład rysunek 2.34, który niedawno oglądałeś. Zadania rozpoznawania spełniają sieci klasyfikujące figury geometryczne, identyfikujące drukowane i ręcznie pisane litery albo sylwetki samolotów czy twarze ludzi.

Jak taka sieć działa?

Aby odpowiedzieć na to pytanie, zacznijmy od sieci skrajnie uproszczonej – to znaczy składającej się z jednego tylko neuronu.

– Że co? Że to właściwie wcale nie jest sieć, bo w sieci powinno być **dużo** neuronów i powinny one być ze sobą **połączone**?

– Nie szkodzi. Okaże się, że nawet taka **namiastka sieci** potrafi się całkiem ciekawie zachowywać!

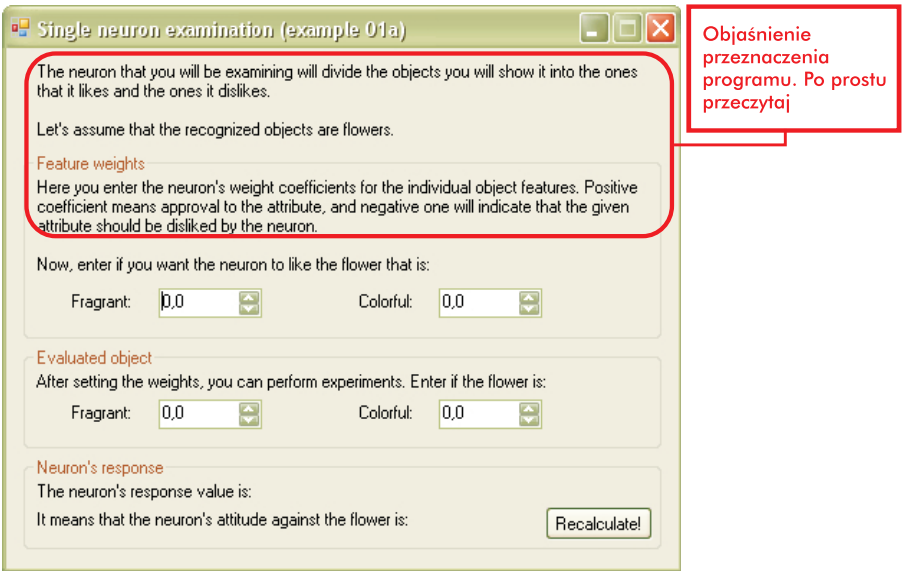
## 4.2. Czego można oczekiwać od pojedynczego neuronu?

Jak zapewne pamiętasz – neuron pobiera sygnały ze swoich wejść, mnoży je przez współczynniki (dla każdego wejścia indywidualnie dobrane w trakcie procesu uczenia) i sumuje tak utworzone iloczyny. Poprzestańmy chwilowo na tym uproszczonym modelu. Wprawdzie dobrze wiesz, że w bardziej rozbudowanych neuronach te zsumowane sygnały są jeszcze przekształcane na sygnał wyjściowy za pośrednictwem odpowiednio dobranej (zwykle nieliniowej) funkcji, ale uwierz mi, że w tym naszym pierwszym uproszczonym neuronie to nie jest konieczne. Zbadamy więc najpierw zachowanie takiego właśnie, prostego, liniowego neuronu. Od czego zależy wielkość jego sygnału wyjściowego?

Otóż łatwo pokazać, że **od stopnia zgodności między sygnałami wejściowymi na poszczególnych wejściach i wartościami współczynników wag na tych wejściach.** Reguła ta jest wprawdzie idealnie dokładna jedynie dla sygnałów wejściowych i wartości wag **znormalizowanych** (za chwilę dokładniej wyjaśnię, jak to należy rozumieć), jednak nawet bez tej precyzyjnej normalizacji – *wielkość sygnału wyjściowego z neuronu może być traktowana jako miara podobieństwa między zbiorem sygnałów wejściowych i zbiorem odpowiadających im wag.*

Można powiedzieć, że neuron ma swoją własną pamięć i w niej przechowuje reprezentację wiedzy, czyli wzorzec wejściowych danych, na które jest „uwrażliwiony" (w postaci aktualnych wartości wag). Jeśli sygnały wejściowe zgodzą się z zapamiętanym wzorcem – neuron „rozpoznaje" je jako coś znajomego i odpowiada na nie silnym sygnałem wyjściowym. Jeśli brak związku między sygnałami podanymi na wejścia neuronu i zapamiętanym wzorcem – sygnał wyjściowy neuronu jest bliski zera (brak rozpoznania). Możliwa jest także sytuacja pojawienia się całkowitej przeciwstawności sygnałów wejściowych i wartości wag – wówczas neuron o liniowej charakterystyce produkuje na swoim wyjściu sygnał **ujemny** – tym silniejszy, im silniej akcentowana jest sprzeczność między „wyobrażeniem" neuronu o zewnętrznym sygnale a jego rzeczywistą wartością.

Na początek zachęcam Cię gorąco żebyś spróbował wykorzystać prosty program o nazwie **Example 01a**. Możesz ten program uruchomić i po prostu wykonać tylko kilka eksperymentów, na które Ci ten program pozwoli, ale możesz skorzystać jeszcze więcej, jeśli zdecydujesz się sam trochę ten program zmienić, uzupełnić i poprawić. Wtedy nauczysz się o sieciach znacznie więcej. Po uruchomieniu programu **Example 01a** zobaczysz okno pokazane na rysunku 4.2, a w jego górnej części tekst objaśniający, co tak właściwie zamierzamy zrobić.



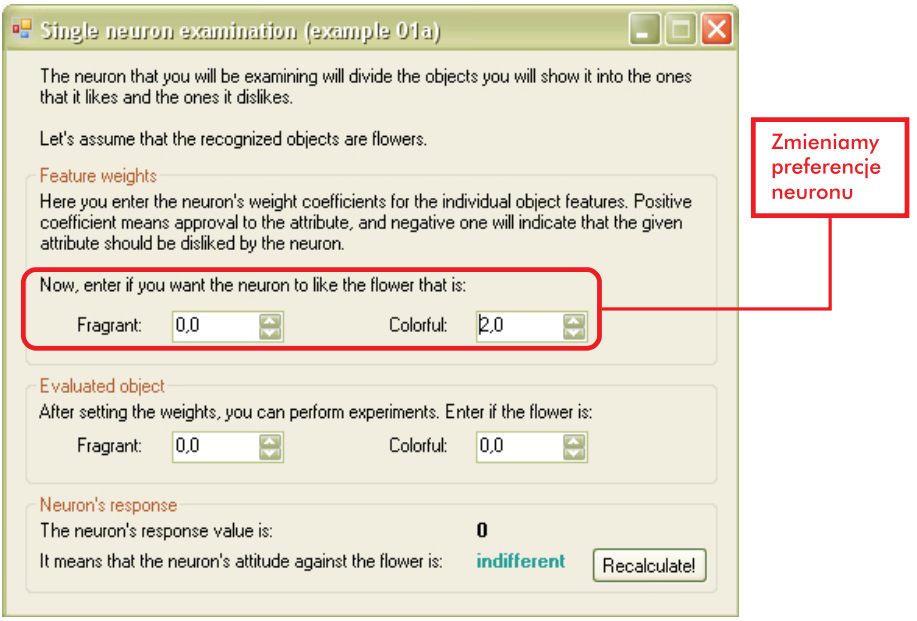

Rys. 4.2. Okno programu **Example 01a** tuż po uruchomieniu

Migający kursor sygnalizuje, że program oczekuje – od Ciebie! – abyś podał wartość współczynnika wagi tego wejścia neuronu, które związane jest z cechą ***zapach*** **(*Fragrant*)**. Możesz ją podać wpisując do pokazanego pola wybraną przez Ciebie liczbę albo możesz zmieniać wartości w okienku, przewijając je klawiszami strzałek w górę i w dół, albo możesz to samo zrobić samą myszą, klikając na przyciski strzałek obok okienka, w którym ustawiasz wartość. Po podaniu odpowiedniej wartości dla wagi cechy ***zapach*** przejdź do następnego pola, które odpowiada za drugą cechę kwiatka – ***barwność*** **(*Colorful*).**

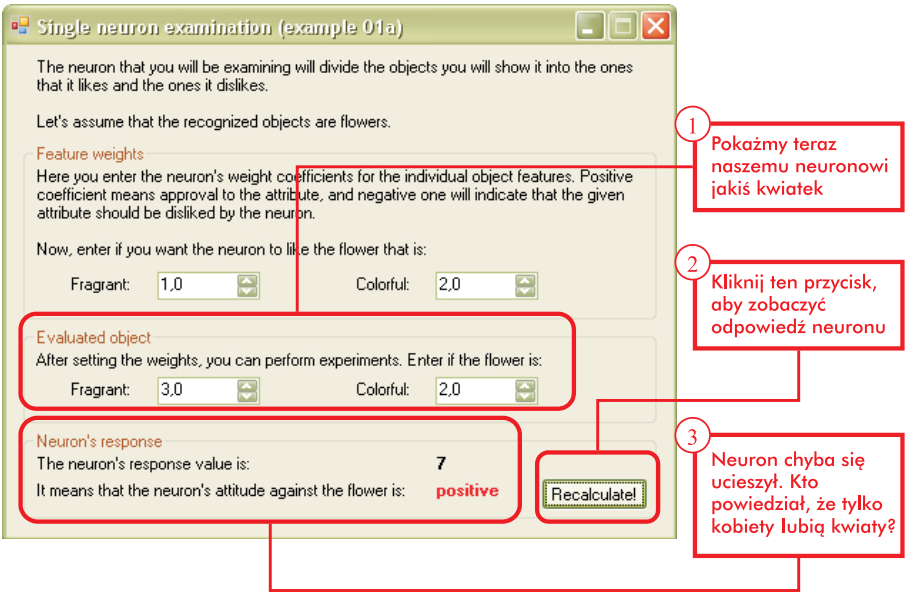
Załóżmy, że chcesz, aby Twój neuron lubił kwiaty kolorowe i pachnące, jednak ze zdecydowaną przewagą dla kolorów. Wówczas po podaniu stosownych odpowiedzi okno programu może tak wyglądać, jak pokazuje rysunek 4.3.

Omawiany program, podobnie jak wszystkie następne, którymi będziesz się posługiwał, daje Ci w każdym momencie możliwość skorygowania Twoich decyzji tak, abyś mógł poprawić wprowadzone dane, zmienić decyzję, skorygować wartości itp. Program stara się na bieżąco uaktualniać wyniki swoich obliczeń.

Po wprowadzeniu danych dotyczących „preferencji" neuronu (czyli – tak naprawdę – wartości występujących w nim współczynników wag) przychodzi czas na sprawdzenie działania neuronu. Możesz podawać różne zestawy składowych wejściowych, w taki sposób, jak pokazano na rysunku 4.4, a pro-




Rys. 4.3. Początkowy etap dialogu użytkownika z programem **Example 01a**

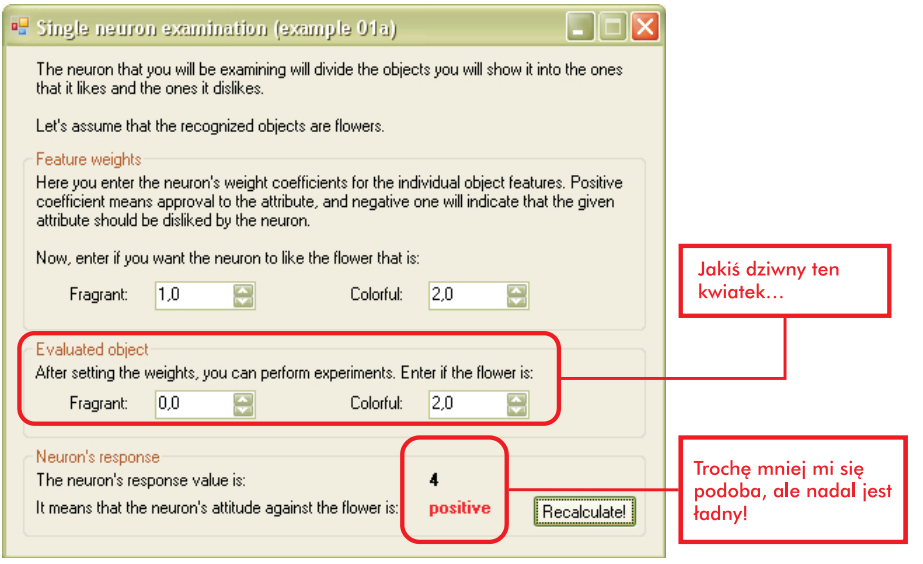



Rys. 4.4. Końcowy etap dialogu użytkownika z programem **Example 01a**

gram obliczy i poda Ci, jaki sygnał wyjściowy wyprodukował neuron i co on oznacza. Pamiętaj, że możesz zmienić zarówno preferencje neuronu, jak i opis kwiatka w dowolnym momencie!

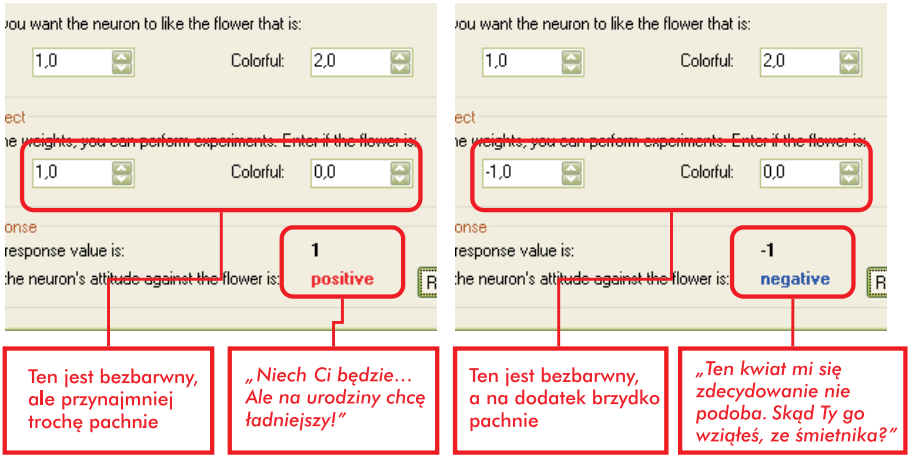
**Uwaga dla dociekliwych:** Jeśli do wprowadzania danych używasz myszki lub klawiszy strzałek, nie musisz – jak zapewne zauważyłeś – klikać przycisku ***Recalculate*** za każdym razem, kiedy chcesz obejrzeć wynik; obliczenia są wykonywane automatycznie. Jeśli jednak wpisujesz liczbę z klawiatury, musisz go kliknąć. To dlatego, że komputer nie wie wtedy, czy to, że od dłuższego czasu nie dotykasz klawiatury, oznacza, że skończyłeś wpisywać liczbę, czy też jesteś w trakcie i tylko sięgnąłeś po kanapkę z tuńczykiem.

Następny etap zabawy będzie polegał na stawianiu naszego neuronu w nietypowych sytuacjach. Istota pokazanego na rysunku 4.5 eksperymentu polegała na tym, żeby sprawdzić, jak neuron zareaguje na pojawienie się obiektu, który będzie odmienny od zapamiętanego przez niego „ideału" (kwiat pachnący i bardzo kolorowy). Pokazaliśmy mu kwiat wprawdzie bardzo kolorowy, ale pozbawiony zapachu. Jak widać – taki również mu się spodobał!



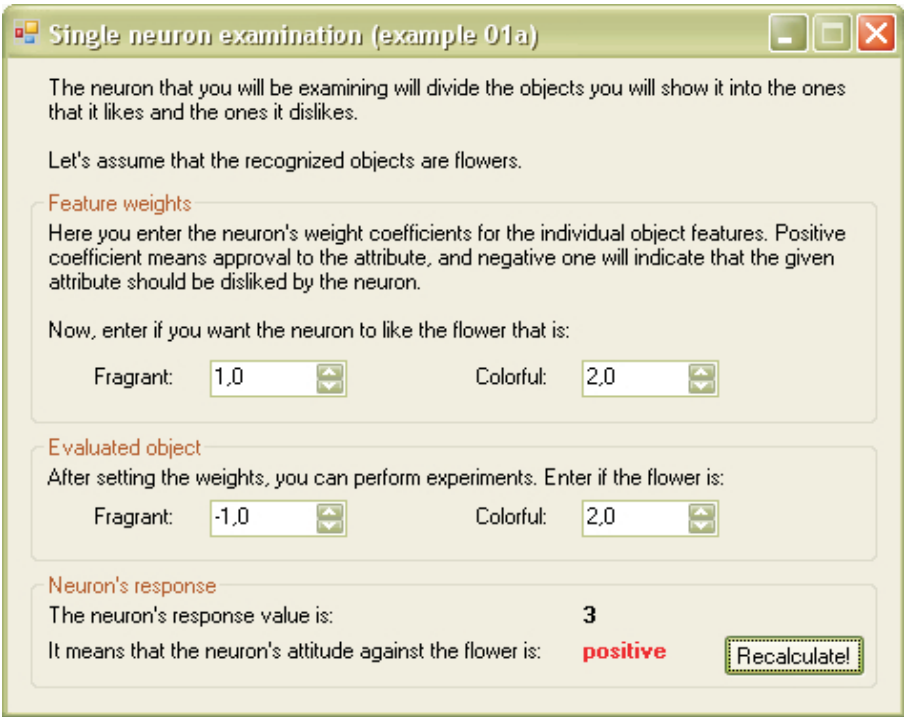

Rys. 4.5. Program **Example 01a** postawiony w nietypowej sytuacji

Zmieniając parametry ocenianego kwiatka, możesz sprawdzać, co neuron zrobi w dowolnych innych okolicznościach. Przykłady takich eksperymentów pokazałem na rysunku 4.6. Sprawdzałem, co neuron zrobi, gdy pokaże mu się kwiatek pachnący, ale praktycznie bezbarwny (jednak go lubi), a potem też mało kolorowy i jeszcze w dodatku brzydko pachnący (takiego to jednak już nie lubi).




Rys. 4.6. Przykład innego eksperymentu wykonywanego z programem **Example 01a**

A teraz postawmy neuronowi zadanie bardziej ambitne: sprawdźmy, czy zaakceptuje on kwiatek brzydko pachnący, jeśli będzie dostatecznie kolorowy.




Rys. 4.7. Jeszcze jeden przykład eksperymentu z programem **Example 01a**

Jak widzisz, możliwości eksperymentowania jest całkiem sporo. Możesz teraz spróbować zmienić preferencje neuronu i zobaczyć, jak w różnych sytuacjach zachowywać się będzie np. neuron, który lubi, gdy kwiatek brzydko pachnie (waga odpowiadająca za zapach powinna być wtedy ujemna).

## 4.3. Co warto zaobserwować podczas dalszych eksperymentów?

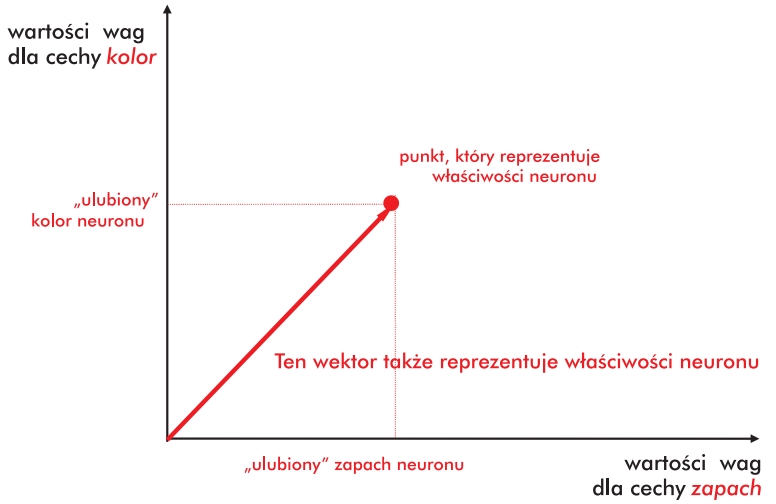
Sugeruję, żebyś poświęcił nieco czasu na to, by poeksperymentować z podanym wyżej programem, gdyż zapewniam Cię, że warto. Podając różne wartości współczynników wag (nie tylko będące liczbami całkowitymi), a także różne wartości wejściowych sygnałów – szybko stwierdzisz, że badany w tej chwili liniowy neuron działa według w miarę prostej zasady. Po prostu traktuje swoje wagi jak „wzorzec" sygnału wejściowego, który chce rozpoznawać. W momencie podania na jego wejście takiej kombinacji sygnałów, która odpowiadała podanym wcześniej wagom – neuron odnajduje w nim coś znajomego i reaguje entuzjastycznie – uzyskujesz więc duży sygnał wyjściowy. W momencie podania innych sygnałów – neuron wykazuje obojętność (małe wartości sygnału wyjściowego) lub przemożny wstręt (ujemne wartości sygnału). No cóż, taka jego natura!

Bliższe badanie pokaże, że zachowanie się neuronu określane jest głównie przez **kąt, jaki tworzy wektor wag z wektorem sygnałów wejściowych.** Obejrzyj to dokładniej, używając kolejnego programu, który także będzie ustalał, które kwiatki neuron lubi, a które nie – tylko że tym razem wzorzec „idealnego kwiatka" będzie pokazany dodatkowo jako punkt (albo wektor) w tak zwanej „przestrzeni wejść".

Pojęcie przestrzeni wejść (oraz związanej z nią „przestrzeni wag") jest dość ważne, a jednocześnie dosyć proste. Celowe jest w związku z tym, byś skupił na tych pojęciach przez chwilę swoją uwagę – zwłaszcza że chodzi o sprawy naprawdę mało skomplikowane, chociaż na pierwszy rzut oka sprawiające wrażenie „wiedzy tajemnej".

Jeśli podajesz poszczególne preferencje neuronu, to znaczy ustalasz, czy „lubi" on (i w jakim stopniu) kwiaty pachnące i (niezależnie od tego) kwiaty kolorowe – to w istocie ustalasz składowe pewnego wektora, tak zwanego **wektora wag**. Możesz więc narysować dwie osie i na jednej z nich (na przykład poziomej) możesz odkładać wartość wagi dla pierwszej cechy (zapach), a na drugiej (na przykład pionowej) wartość wagi dla drugiej cechy (rys. 4.8). Jeśli będziesz chciał zaznaczyć preferencje konkretnego neuronu – możesz odmierzyć odpowiednie położenie na osi poziomej i na osi pionowej, a punkt wyznaczony przez te dwie współrzędne będzie odpowiadał rozważanemu kon-

kretnemu neuronowi. Na przykład neuron, który „ceni" tylko zapach kwiatu, a kolor „jest mu raczej obojętny" – będzie reprezentowany przez punkt położony maksymalnie na prawo (duża wartość pierwszej składowej), ale leżący na poziomej osi układu współrzędnych (zerowa lub inna niewielka wartość drugiej składowej). Z kolei neuron, któremu chcesz przypisać zamiłowanie do storczyków – kwiatów o pięknych barwach, lecz o mizernym lub czasem wręcz przykrym zapachu – będzie leżał dość wysoko na osi pionowej (wysoka ocena waloru barwności) lub nawet na lewo od tej osi (zgoda na akceptację niemiłego zapachu).

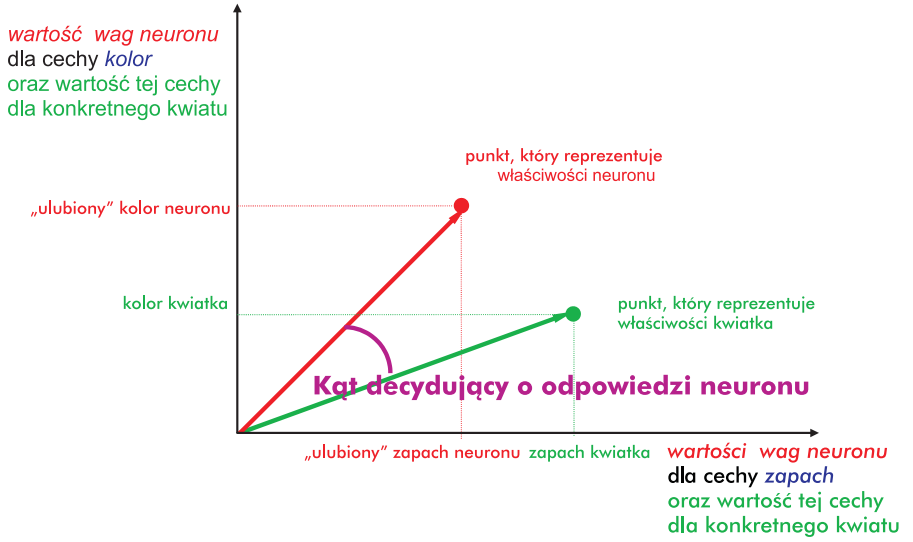



Rys. 4.8. Reprezentacja właściwości neuronu jako punktu i jako wektora w przestrzeni cech

Identycznie możesz postąpić z każdym konkretnym obiektem (kwiatem), który „przedstawiasz" neuronowi do oceny: jego barwę odłożysz na osi pionowej, a zapach na osi poziomej. Jeśli będzie to konwalia – to reprezentować ją będzie punkt położony daleko na prawo wzdłuż osi poziomej (trudno o piękniejszy zapach), ale zdecydowanie nisko (kolor nie jest walorem tego kwiatka), zaś jeśli zdecydujesz się na pokazanie goździka – pojawi się on wysoko (ma z reguły ładny kolor), ale raczej po ujemnej stronie osi poziomej (zapach goździków trudno uznać za przyjemny). W lewym dolnym rogu układu współrzędnych spotkasz rosiczkę (to taka owadożerna roślina, która przywabia muchy wyglądem przypominającym gnijące mięso i takimż zapachem), zaś majestatyczne róże zajmą zapewne prawy górny narożnik.

Podczas rozważań dotyczących sieci neuronowych wygodnie jest określone obiekty (konkretne oceniane kwiaty, a także „wyobrażenie" neuronu o tym, jak wygląda idealny kwiat) zaznaczać nie tylko w postaci punktów w przestrzeni sygnałów, ale w postaci **wektorów.** Potrzebne wektory uzyskasz łącząc na wykresie zaznaczany punkt z początkiem układu współrzędnych. Tak właśnie będzie postępował program **Example 01b,** który – podobnie jak poprzedni program – znajdziesz w swoim menu **Start**[1]. Bawiąc się tym programem, zwróć uwagę na następujące fakty:

⇒ wielkość sygnału wyjściowego neuronu zależy głównie od **kąta** pomiędzy wektorem wejściowym (reprezentującym sygnały wejściowe) a wek-




Rys. 4.9. Wzajemne położenie wektora wag i wektora sygnałów wejściowych – jako czynnik determinujący wielkość sygnału wyjściowego neuronu

torem wag („idealnym wzorcem" aprobowanym przez neuron). Ilustruje to rysunek 4.9;

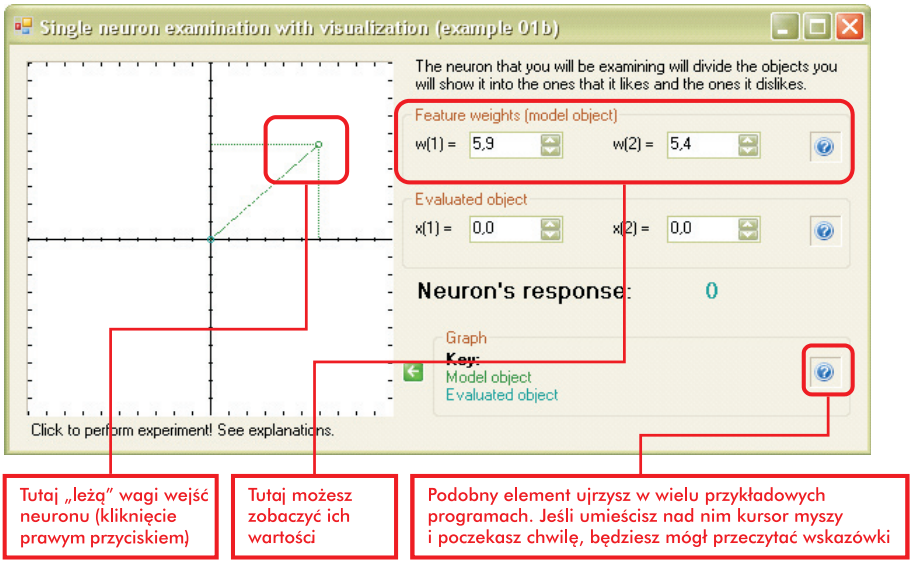
⇒ jeśli kąt między wektorem wejściowym a wektorem wag jest **mały** (oba wektory leżą blisko siebie) – neuron daje silny sygnał **pozytywny;**

⇒ jeśli kąt między wektorem wejściowym a wektorem wag jest **duży** (oba wektory tworzą kąt zdecydowanie większy od 90°) – neuron daje silny sygnał **negatywny;**

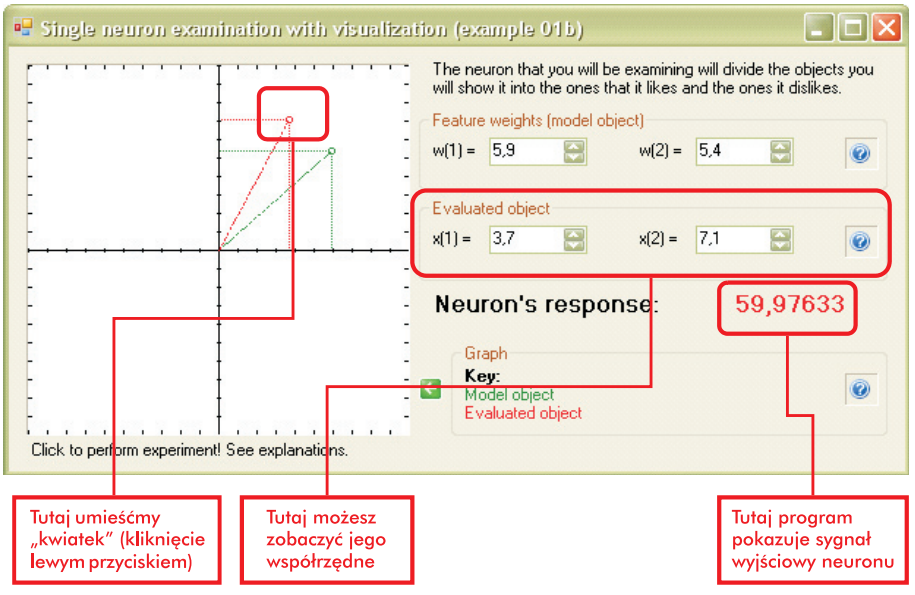
⇒ jeśli kąt między wektorem wejściowym a wektorem wag jest **bliski 90°** – neuron daje słaby sygnał **neutralny** (bliski 0);

⇒ jeśli długość wektora wejściowego jest znacząco mniejsza od długości

[1] Jeśli jeszcze nie zainstalowałeś przykładowych programów, zrób to koniecznie teraz – zabawa dopiero się rozkręca…

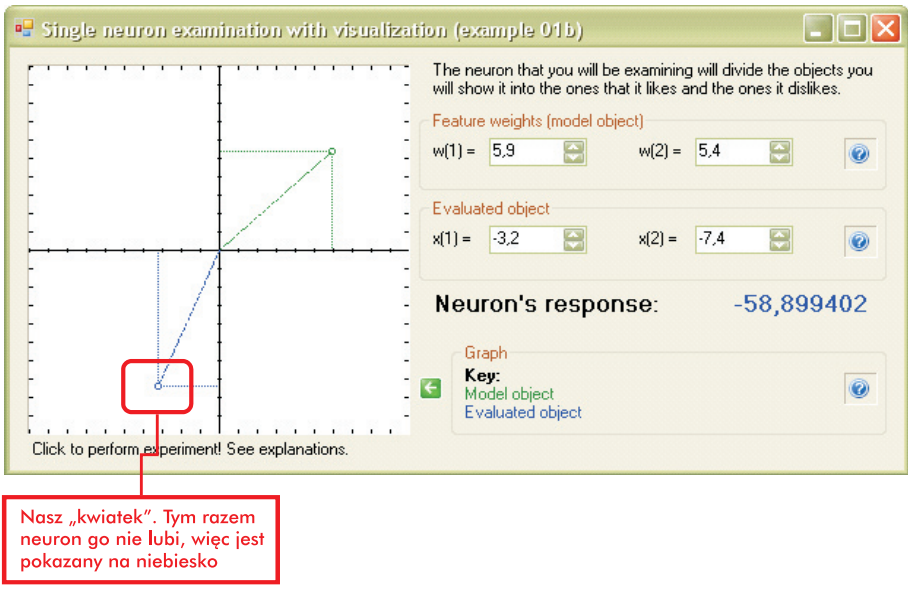



Rys. 4.10. Okno programu **Example 01b** z zaznaczonym obiektem wzorcowym

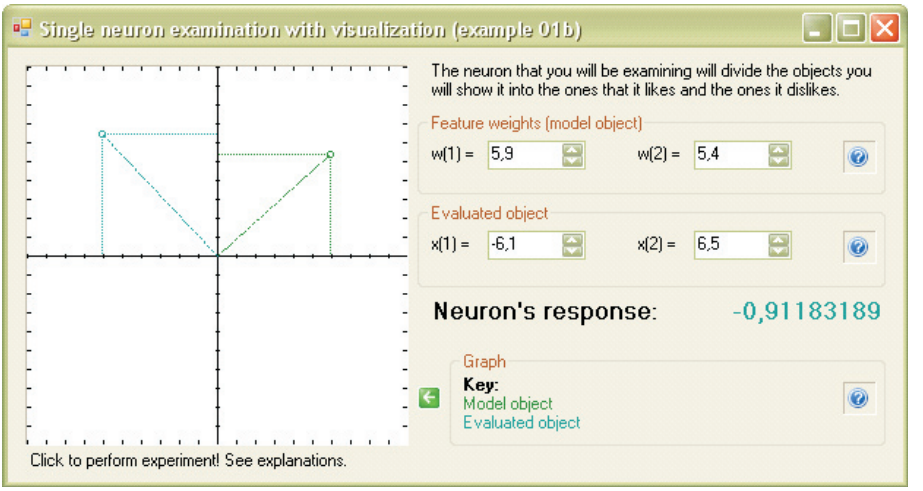



Rys. 4.11. Prezentacja położenia wektora wejściowego dla którego sygnał wyjściowy jest dodatni

wektora wag – neuron daje sygnał **neutralny** (bliski 0) niezależnie od kierunku wektora wejściowego**.**




Rys. 4.12. Prezentacja położenia wektora wejściowego, dla którego sygnał wyjściowy jest ujemny




Rys. 4.13. Prezentacja położenia wektora wejściowego, dla którego sygnał wyjściowy jest obojętny

Wszystkie opisane wyżej właściwości i charakterystyczne cechy zachowania neuronu możesz sam dokładnie przebadać korzystając z programu **Example 01b.** Wprawdzie rysunki produkowane przez ten program nie są tak dopracowane graficznie, jak rysunek 4.9, ale są na tyle czytelne, że powinny być wygodną

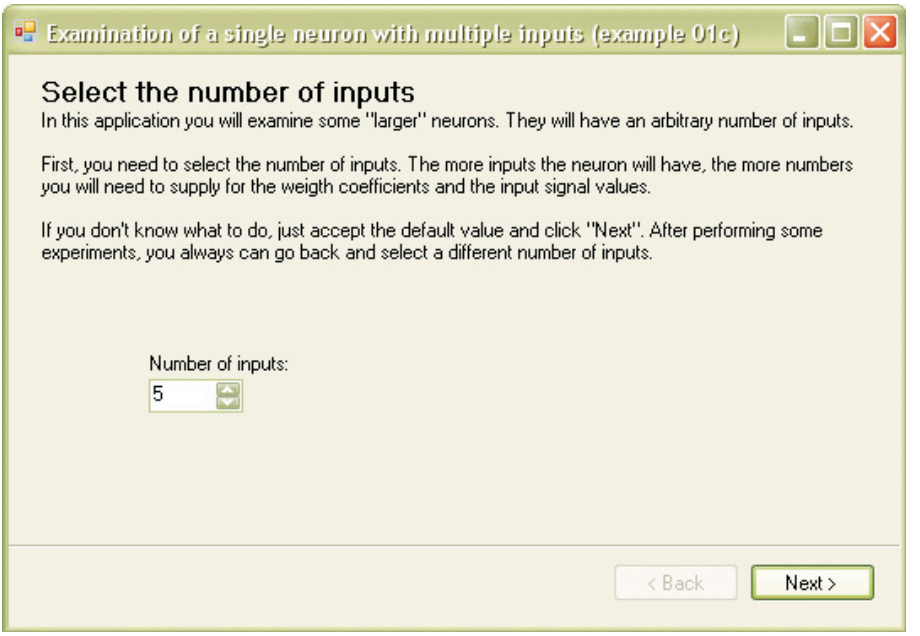
podstawą do zbierania przez Ciebie własnych doświadczeń, pozwalających namacalnie poznać i zapamiętać, co właściwie robi taki pojedynczy neuron.

Obsługa programu jest bardzo intuicyjna. Jedyne, co musisz robić, to klikać na obszarze widocznego po lewej stronie wykresu. Kliknij najpierw **prawym** przyciskiem myszy, aby ustalić położenie wybranego przez Ciebie punktu, odpowiadającego pewnym współczynnikom wag neuronów (patrz rys. 4.10). Program pokaże Ci ten punkt oraz jego współrzędne. Oczywiście, możesz je w każdym momencie zmienić, klikając ponownie na wykresie w innym miejscu lub ręcznie modyfikując jego współrzędne – tak jak robiłeś to przy programie **Example 01a**. Teraz kliknij **lewym** przyciskiem na wykresie, aby ustalić położenie „kwiatka", i obserwuj odpowiedź programu. Jeśli neuronowi dany kwiatek się podoba (po prawej stronie podawana jest wartość sygnału wyjściowego, więc możesz się łatwo przekonać, co neuron „sądzi" o Twoim kwiatku), wówczas odpowiedni punkt oznaczony jest na wykresie kolorem **czerwonym** (jak szczyty gór na mapie – patrz rys. 4.11). Jeśli stosunek jest negatywny – punkt oznaczany jest kolorem **błękitnym** (jak dno oceanu na mapie – patrz rys. 4.12). Jeśli natomiast reakcja neuronu jest obojętna – odpowiedni punkt oznaczany jest na wykresie jako **bladoniebieski** (rys. 4.13). Po pewnym czasie zdołasz zapewne wytworzyć sobie pogląd na temat tego, jak wyglądają obszary odpowiadające poszczególnym decyzjom w przestrzeni sygnałów wejściowych. Aby łatwiej je sobie wyobrazić, możesz spróbować „ciągnąć" myszą po wykresie z wciśniętym którymś z przycisków, aby obserwować płynnie zmieniające się wyniki.

## 4.4. Jak sobie poradzić z większą liczbą wejść neuronu?

Przytoczone wyżej przykłady były sympatycznie proste, ponieważ dotyczyły **jednego** neuronu o **dwóch** zaledwie sygnałach wejściowych. W rzeczywistych zastosowaniach sieci neuronowych masz jednak zwykle do czynienia z zadaniami, w których rozważa się wiele wejść (bo poszukiwane rozwiązanie, które próbujesz znaleźć z użyciem sieci neuronowych, zależeć będzie na ogół od dużej liczby różnych danych wejściowych), a także trzeba zastosować wiele współpracujących ze sobą neuronów, żeby uzyskać taki stopień złożoności obliczeń realizowanych przez sieć, jaki jest potrzebny z punktu widzenia rozważanego zadania.

To jest właśnie to miejsce, w którym w sieciach neuronowych „kończą się żarty, a zaczynają się schody". Narysować się tego nie da (bo punktów w przestrzeni wejść mającej kilkanaście składowych nikt sobie wyobrazić nie zdoła!) i dlatego wszystkim się wydaje, że jest to bardzo trudne. Tymcza-



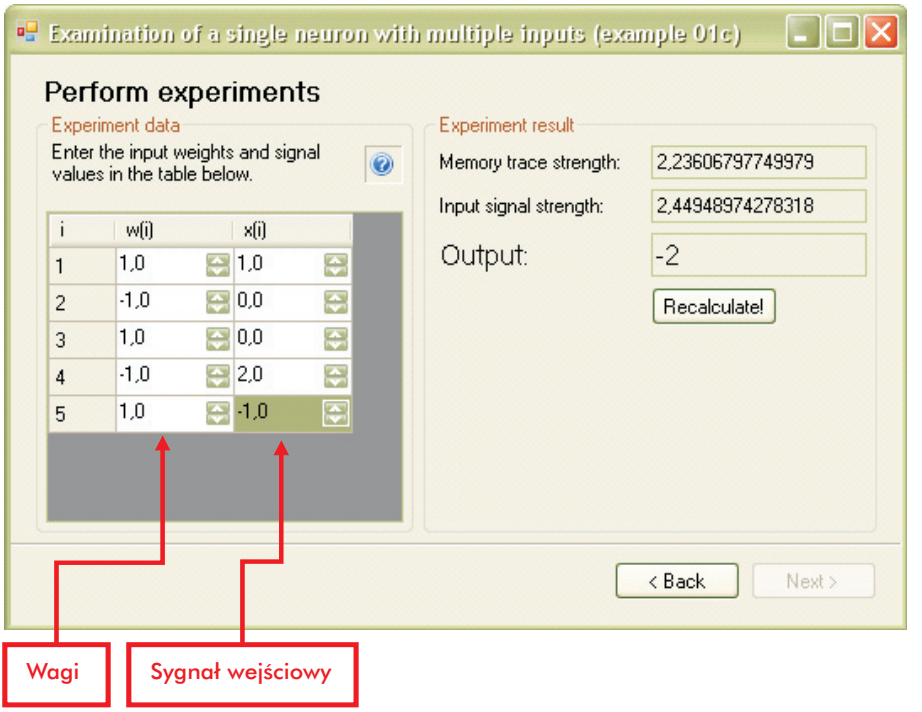

Rys. 4.14. Początek konwersacji z programem **Example 01c**

sem wcale tak być nie musi. Zaraz Cię o tym przekonam. Wypróbuj program **Example 01c** (patrz rys. 4.14).

Program ten najpierw prosi, abyś podał ilość wejść do neuronu. Możesz zaakceptować domyślną wartość (5) i kliknąć **Next**. Następnie, podobnie jak wcześniej przy programie **Example 01**, powinieneś podać współczynniki wag, czyli wzorzec sygnału, na który Twój neuron ma pozytywnie reagować. Umieść je w kolumnie oznaczonej **w(i)** (rys. 4.15). Potem podajesz wartości sygnałów wejściowych w kolumnie **x(i)** – a program oblicza i podaje Ci wartość sygnału wyjściowego. Prawda, że proste?

Nie przejmuj się, że program coś majaczy o jakichś mocach (może chodzi o moce piekielne?). Za chwilę dowiesz się, do czego to może Ci się przydać.

Podczas eksperymentów z tym programem możesz łatwo zauważyć, że mimo poruszania się w **pięciowymiarowej** przestrzeni wag i sygnałów (taką przestrzenią nawet Albert Einstein się nie zajmował, poprzestając na czterowymiarowej czasoprzestrzeni!) – możesz stosunkowo łatwo przewidzieć, co neuron zrobi. Jeśli podając kolejne współrzędne wektora sygnałów wejściowych ustawisz je tak, że będą one podobne do odpowiadających im współrzędnych wektora wag (współrzędne te program usłużnie przypomina podczas wprowadzania sygnałów wejściowych) – uzyskasz dużą wartość syg-




Rys. 4.15. Dalsza część konwersacji z programem **Example 01c**

nału wyjściowego z neuronu. Jest to uzasadnione i dość oczywiste: podobne wartości składowych wag i tych samych składowych sygnałów oznaczają, że wektor wag leży blisko wektora sygnałów, a to – jak już wiesz z wcześniejszych rozważań – jest warunkiem uzyskania dużej wartości sygnału wyjściowego. Z kolei, jeśli wartości poszczególnych składowych wektora sygnałów wejściowych przyjmiesz jako bliskie odpowiednim składowym wektora wag, **ale z przeciwnym znakiem** – spowodujesz w efekcie, że wektor sygnałów wejściowych będzie skierowany w przeciwną stronę niż wektor wag. Efektem będzie oczywiście to, że neuron wyśle silny **ujemny** sygnał jako swoją odpowiedź na takie pobudzenie. Na koniec, jeśli dobierając sygnały wejściowe nie będziesz starał się zachować jakiejkolwiek zgodności ze składowymi wektora wag – masz duże szanse, że Twój wielowymiarowy neuron zachowa wobec tych sygnałów wyniosłą obojętność, produkując na swoim wyjściu bardzo niewielki (co do wartości bezwzględnej) sygnał wyjściowy.

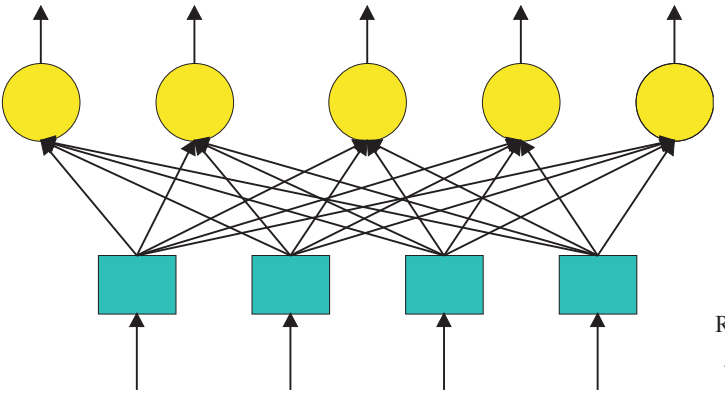
Przy okazji prowadzonych eksperymentów zauważysz zapewne jedną prawidłowość. Otóż duże wartości sygnału wyjściowego neuronu można uzyskać na dwa sposoby: albo poprzez podanie na jego wejścia sygnałów **zgodnych z odpowiednimi wagami na tych wejściach** (tego właśnie oczekiwaliśmy),

albo po prostu poprzez podanie **dużych** sygnałów na wejścia o dodatnich wartościach wag. Pierwszy sposób uzyskania dużej wartości sygnału wyjściowego jest inteligentny, wytworny i elegancki – cóż, kiedy te same (a czasem nawet lepsze efekty) można uzyskać stosując brutalną siłę, czyli wykorzystując sposób numer dwa. Dlatego wprowadzając sygnały wejściowe powinieneś starać się, by były one zawsze mniej więcej tej samej „mocy" (wykorzystując podawany przez komputer parametr oszacowujący aktualną „moc" sygnału). Tylko wtedy będziesz mógł poprawnie interpretować i porównywać wyniki. Podobnie porównując zachowanie różnych neuronów (o różnych wektorach wag) dla takich samych sygnałów wejściowych – powinieneś dążyć do tego, by miały one podobną wartość „mocy śladu pamięciowego" – czyli po prostu długości wektora wag. W sieci zawierającej większą liczbę neuronów znaczenie „mocy" sygnałów wejściowych radykalnie zmaleje, bo te same – silniejsze lub słabsze sygnały – doprowadzane będą do wszystkich neuronów, a o efekcie zadecyduje **różnica** sygnałów z różnych neuronów: tych lepiej „dostrojonych" do podawanego sygnału i tych dostrojonych gorzej. Porozmawiamy o tym za chwilę nieco dokładniej. Jednak w przypadku rozważania tylko jednego neuronu sprawa niejednakowych wielkości sygnałów wejściowych może trochę psuć obraz i utrudniać interpretację wyników – dlatego umówmy się: wybierając sygnały wejściowe dla badanego neuronu staraj się przyjmować takie ich wartości, by ich suma kwadratów miała (w przybliżeniu, duża dokładność nie jest tu potrzebna) jakąś raz na zawsze ustaloną wartość – na przykład między **5** a **35.** Ponieważ program oblicza moc jako pierwiastek z sumy kwadratów współrzędnych (bo taki jest wzór na długość wektora), umówmy się, że moc sygnałów powinna zawierać się między pierwiastkiem z 5 a pierwiastkiem z 35, a więc gdzieś między 2 a 6.

Dlaczego akurat tyle? Bo przekonałem się podczas własnych eksperymentów z programem, że przy podaniu niewielkich przypadkowych wartości całkowitych dla pięciu wejść neuronu – właśnie takie (mniej więcej) wartości zwykle dostaniesz. Jednak jeśli wolisz, możesz przyjąć dowolną inną ustaloną wartość i też będzie dobrze, bylebyś się jej potem konsekwentnie trzymał. To samo ograniczenie przyjmij odnośnie wartości współczynników wag (przyda Ci się to na przyszłość, zobaczysz!), dzięki czemu łatwo będzie sprawdzać, na ile wprowadzane sygnały wejściowe „przystają" do wartości wag.

## 4.5. Jak się zachowuje prosta liniowa sieć neuronowa?

Załóżmy teraz, że neuronów jest więcej i tworzą one **sieć.** Przyjmiemy chwilowo, że sieć jest **jednowarstwowa** (to znaczy neurony nie łączą się między sobą, a tylko z wejściem i z wyjściem – patrz rys. 4.16).



Rys. 4.16. Struktura jednowarstwowej sieci neuronowej

Sygnały wejściowe, które będziesz dostarczał, będą więc podawane bezpośrednio na wejścia wszystkich neuronów, a sygnały wyjściowe tych neuronów będą z kolei traktowane jako odpowiedź całej sieci na postawione zadanie.

Jak taka sieć działa?

Każdy z neuronów ma swój własny zestaw współczynników wagowych, jest więc gotowy do rozpoznawania jakiegoś charakterystycznego zestawu sygnałów wejściowych, przy czym każdy neuron ma oczywiście inny taki wzorzec, który rozpoznaje. Kiedy więc podasz do sieci jakieś sygnały wejściowe – każdy neuron wyznaczy **niezależnie** od pozostałych swój sygnał wyjściowy, który okaże się duży dla jednego neuronu (bo rozpoznał on „swój" wzorzec) i mniejszy dla wszystkich innych. Obserwując sygnały wyjściowe neuronów, możesz się łatwo zorientować, jaki wzorzec sieć „sugeruje" (na podstawie tego, który neuron wysyła sygnał o największej wartości), a także możesz ocenić, na ile sieć jest „pewna" swojej decyzji – porównując sygnał wyjściowy ze „zwycięskiego" neuronu z sygnałami generowanymi przez pozostałe elementy sieci.

Czasem ta właśnie cecha sieci, pozwalająca na wykrycie sytuacji niepewnych i niejednoznacznych, bywa najbardziej użyteczna w praktyce, nie ma bowiem nic gorszego niż algorytm ferujący bardzo zdecydowane i jednoznaczne opinie w oparciu o niepełne dane i przybliżone metody wnioskowania. Jeśli taki algorytm jest rozsądnie używany – może być oczywiście także użyteczny. Jednak ludzie zwykli żywić tym większe zaufanie do komputerów, im mniej je rozumieją. Jeśli więc wyobrazimy sobie głupca wyposażonego w nazbyt arbitralnie działający system ekspertowy – to wizja małpy z brzytwą jest wobec takiego horroru dziecinną igraszką...

## 4.6. Jak zbudować prostą liniową sieć neuronową?

Przedstawię Ci teraz kolejny prosty program. Program ten, nazwany **Example 02**, pomoże Ci w rozpoczęciu prostej zabawy z bardzo małą siecią neuronową, zachęcam Cię jednak gorąco, żebyś potem na podobnej zasadzie sam napisał jakiś większy program, rozwiązujący któryś z Twoich prawdziwych problemów.

Opisana w programie **Example 02** sieć rozpoznaje zwierzęta. Wyróżnione są trzy kategorie zwierząt **(ssak, ryba, ptak**), dlatego sieć zawiera trzy neurony. Rozpoznawanie dokonywane jest na podstawie 5 cech, w związku z tym każdy neuron ma pięć wejść. Na wejścia te podawane są następujące informacje:

⇒ ile zwierzę ma nóg,
⇒ czy żyje w wodzie, czy umie latać,
⇒ czy jest pokryte piórami, oraz
⇒ czy rodzi się z jaj.

Dla poszczególnych neuronów ustaliłem od razu wartości wag, w taki sposób, żeby zawarte w nich były wzorce odpowiednich zwierząt. W związku z tym dla neuronu numer 1, który ma rozpoznawać **ssaka**, podałem następujące wartości wag kolejnych wejść:

⇒ 4 = ssak ma 4 nogi,
⇒ 0.01 = ssak czasem żyje w wodzie (foka), chociaż nie jest to dla niego typowe,
⇒ 0.01 = ssak czasem umie latać (nietoperz), chociaż nie jest to dla niego typowe,
⇒ –1 = ssak nie ma piór,
⇒ –1.5 = ssak jest żyworodny i jest to jego ważna cecha.

Odpowiednio dla drugiego neuronu rozpoznającego **ptaka** ustaliłem wagi:

⇒ 2 = ptak ma dwie nogi,
⇒ –1 = ptak nie żyje w wodzie (kaczka pływa tylko na powierzchni!),
⇒ 2 = ptak zwykle – co jest dla niego ważne – umie latać (wyjątek: struś),
⇒ 2.5 = ptak ma pióra i jest to jego ważna cecha,
⇒ 2 = ptak jest jajorodny.

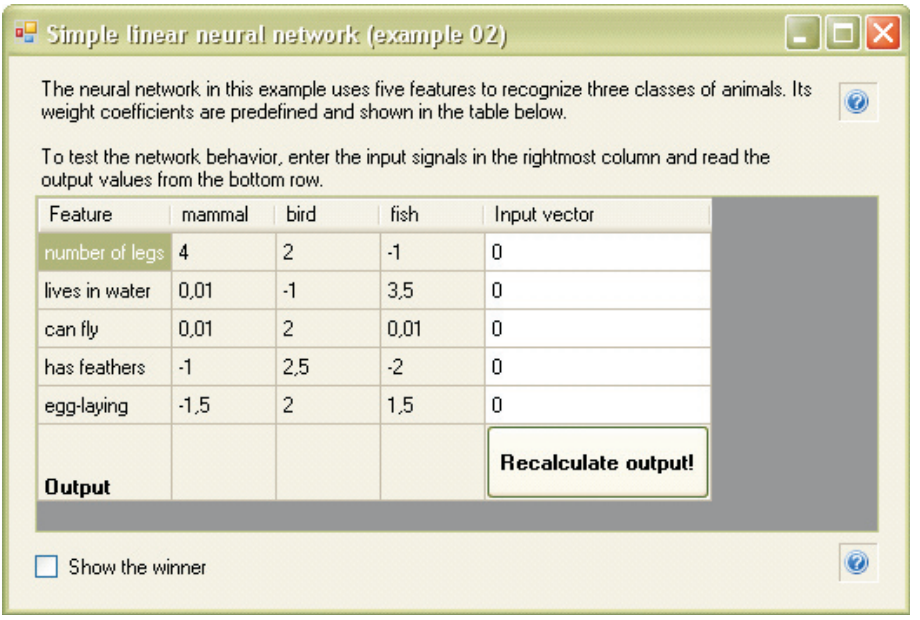
I podobnie dla trzeciego neuronu identyfikującego **rybę** wybrałem wagi:

⇒ –1 = ryba nie ma nóg,
⇒ 3.5 = ryba żyje w wodzie i jest to jej najważniejsza cecha,
⇒ 0.01 = ryba **na ogół** nie umie latać (chociaż istnieją ryby latające!),

⇒ –2 = ryba nie jest **nigdy** pokryta piórami ani niczym, co je przypomina,

⇒ 1.5 = ryba **zwykle** jest jajorodna, co jednak nie jest tak ważną cechą, jak w przypadku ptaka, bo są wyjątki (ryby żyworodne).

Program po uruchomieniu go podaje informację o podanych wyżej wartościach współczynników wagowych dla wszystkich wejść wszystkich neuronów (rys. 4.17) i pozwala Ci na prowadzenie całkiem zabawnych eksperymentów, które opiszę w kolejnym podrozdziale.




Rys. 4.17. Początek pracy programu **Example 02**

## 4.7. Jak wykorzystywać opisaną sieć neuronową?

Jak już wspomniałem, przedstawiony wyżej program zakłada, że sieć ma trzy wyjścia, skojarzone z rozpoznawaniem trzech rodzajów obiektów (zwierząt): *ssaków, ptaków i ryb.* Ponieważ sieć jest jednowarstwowa – oznacza to, że zawiera ona **tylko** te trzy neurony, chociaż w przyszłości poznasz sieci, w których liczba neuronów będzie znacznie większa niż liczba wyjść. Ze względu na prostą budowę opisanej sieci bardzo łatwo możesz ją sam rozbudować do dowolnie dużej liczby wyjść.

Na wejście sieci podaje się – w rozważanym przykładzie – tylko pięć sygnałów, odpowiadających pewnym cechom rozpoznawanych obiektów.

Oczywiście, w banalny sposób możesz tę liczbę powiększyć, jeśli w zadaniu, które Ty będziesz chciał powierzyć rozważanej sieci, będzie więcej danych wejściowych. Wszystkie sygnały wejściowe są doprowadzane do wszystkich neuronów, co jest zgodne z „zasadą lenia": skoro nie chce Ci się myśleć, które sygnały wejściowe mają wpływ na które wyjścia z sieci – to najlepiej wybrać połączenia na zasadzie „każdy z każdym". W praktyce w sieciach neuronowych zawsze tak się postępuje, warto więc, żebyś już się zaczął do tej konwencji przyzwyczajać.

Dobrze by było także, gdybyś się odrobinę zastanowił nad tym, **jak będziesz podawał sygnały wejściowe do sieci**. Są wśród nich sygnały o charakterze liczbowym (informacja, ile dane zwierzę ma nóg), a także sygnały odwzorowujące cechy o charakterze logicznym (czy dane zwierzę żyje w wodzie, czy umie latać, czy jest pokryte piórami, a także czy jest jajorodne). Odnośnie do tych ostatnich sygnałów trzeba się jakoś umówić, w jaki sposób będą one reprezentowane w sieci, bo neurony operują – jak już kiedyś z naciskiem pisałem – **wartościami** sygnałów, a nie symbolami (takimi, jak ***prawda*** i ***fałsz***). Ponieważ sprawa nie jest oczywista – pozwól, że podzielę się z Tobą pewną refleksją.

Jeśli jesteś informatykiem (mam podejrzenia, że jesteś, skoro interesujesz się sieciami neuronowymi), to możesz żywić podejrzenia, że pojęcia **prawdy** i **fałszu** da się znakomicie zapisać w formie binarnej: **1** to prawda, a **0** to fałsz. Jeśli zaś jesteś Znakomitym Informatykiem, zwłaszcza takim, który grzebie w Asemblerze, i co noc śnią Ci się rejestry mikroprocesora, heksadecymalne zapisy pamięci oraz aplety Javy – to takie właśnie przyporządkowanie prawdy i fałszu jest dla Ciebie oczywiste, **jedynie** słuszne i absolutnie prawidłowe.

Hmmm...

Muszę Ci coś tu wyznać. Otóż wchodząc w dziedzinę sieci neuronowych będziesz musiał pewne swoje wcześniejsze przyzwyczajenia nieco zmodyfikować. Tutaj spotkasz się z tym po raz pierwszy, ale nie po raz ostatni!

Zapamiętaj więc, że **zero** jest w sieci neuronowej dość głupim sygnałem (takim, którego nie ma, więc nie wnosi żadnej informacji). Wynika to ze sposobu działania neuronów, które mnożą sygnały przez wagi i dodają je – a mnożenie przez zero daje zawsze taki sam i w dodatku mało ciekawy wynik – niezależnie od wartości wagi, która jest przecież wyrazem wewnętrznej wiedzy neuronu!

Decydując się na użycie zera jako sygnału wejściowego, sam pozbawiasz się pewnych możliwości wpływania na zachowanie sieci, co nie jest mądre ani potrzebne. Dlatego w moim programie zastosowałem konwencję, do której **stałego** używania w sieciach neuronowych gorąco Cię namawiam: prawda

oznaczana jest sygnałem **+1**, zaś nieprawda sygnałem **–1**. Takie „bipolarne" sygnały naprawdę lepiej spełniają swoje zadanie!

Dodatkową zaletą bipolarnej sieci neuronowej jest możliwość używania (podczas wprowadzania danych) **dowolnych** wartości sygnałów wejściowych, które mogą odzwierciedlać przekonanie osoby korzystającej z sieci o ważności takich lub innych wiadomości. Możesz więc wprowadzając dane dla **dorsza** uznać za istotny fakt, że żyje on w wodzie i na odpowiednie wejście podać sygnał +2 zamiast +1 (można to interpretować jako okrzyk „*ależ tak, oczywiście!*" zamiast chłodnego „*tak, to prawda"*), a w innych przypadkach możesz podawać wartości mniejsze od jedności (na przykład wprowadzając odpowiedź na pytanie „czy umie latać" dla ryby latającej możesz mieć wątpliwości, czy powinieneś udzielić odpowiedzi „+1", tak jak w przypadku orła – możesz więc podać jakieś niepewne i pełne wahania „+0,2" , co daje się interpretować jako *„trochę..."*). Możliwości jest tu oczywiście znacznie więcej, na przykład, gdyby był sygnał odpowiadający pojęciu *„ma ogon",* to dla węża należałoby wprowadzić wartość +10 (*„wyłącznie!"*).

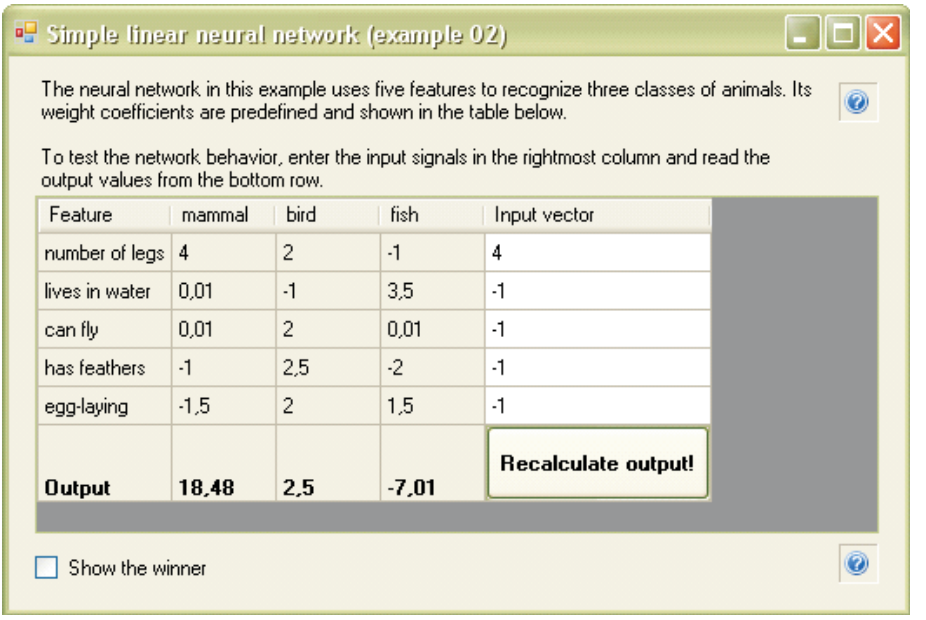
Skoro już wiesz, jak podawać sygnały do sieci – spróbuj wraz ze mną wykonać kilka prostych eksperymentów. Podaj dane dla kilku dowolnie wybranych zwierząt i sprawdź, czy sieć poprawnie je rozpozna. Możesz przy tym zauważyć, że w sieci składającej się z wielu elementów mniejsze znaczenie ma kwestia normalizacji wejściowych sygnałów (zabieganie o to, by miały zawsze tę samą wypadkową „moc"), ponieważ do takich samych wniosków dojdziesz, podając – przykładowo – dla **lisa** wartości:

⇒ 4 (nogi),
⇒ –1 (nie wodny),
⇒ –1 (nie lata),
⇒ –0,9 (nie jest pokryty piórami – chyba że akurat złapał gęś...),
⇒ –1 (nie jajorodny);

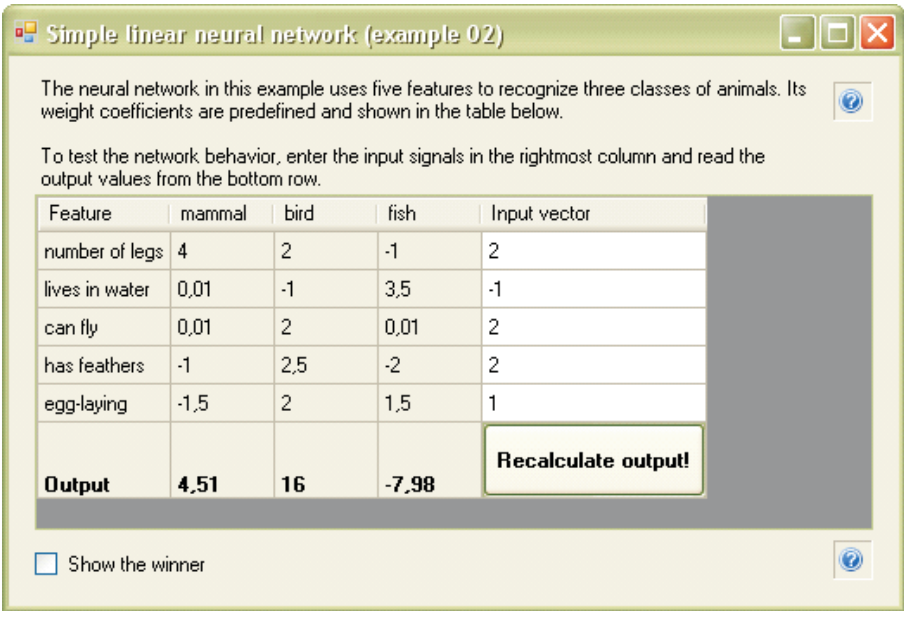
jak i wartości:

⇒ 8 (ma 4 nogi ale jest to tak ważne, że warto podać dwa razy),
⇒ –6 (nie cierpi wody, zwłaszcza we wkopanych beczkach!),
⇒ –3 (absolutnie nie lata, chociaż czasem by chciał...),
⇒ –5 (rude futro to ważny znak rozpoznawczy),
⇒ –9 (nie składa jajek i wyprasza sobie podobne podejrzenia!).

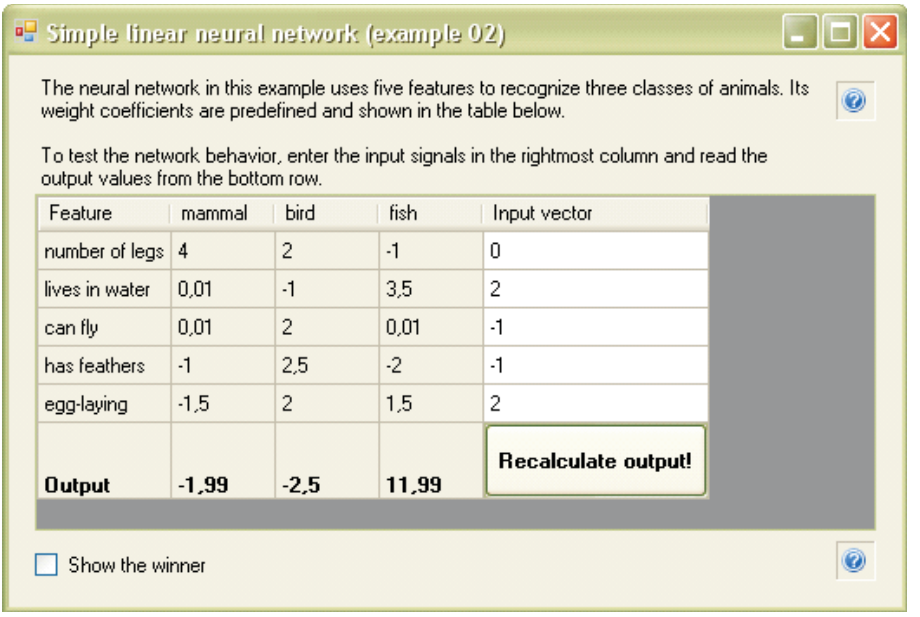
Warto także zauważyć, bawiąc się zaproponowanym programem, że nie tylko potrafi on poprawnie rozpoznawać typowe sytuacje (prawidłowo zidentyfikuje każdego typowego ssaka, ptaka i rybę – patrz rys. 4.18, 4.19 i 4.20), ale dość sensownie zachowuje się także w odniesieniu do sytuacji nietypowych – na przykład rozpozna jako ssaka fokę, nietoperza, a nawet dziobaka (taki dziwny australijski ssak, który jest jajorodny), a jako ptaka – na przykład




Rys. 4.18. Rozpoznawanie przy pomocy sieci typowego ssaka




Rys. 4.19. Działanie programu podczas identyfikacji ptaka




Rys. 4.20. Działanie programu, podczas gdy rozpoznaje on rybę

nie latającego strusia. Nie straszne są mu także ryby latające, które także rozpoznaje „bez pudła". Spróbuj!

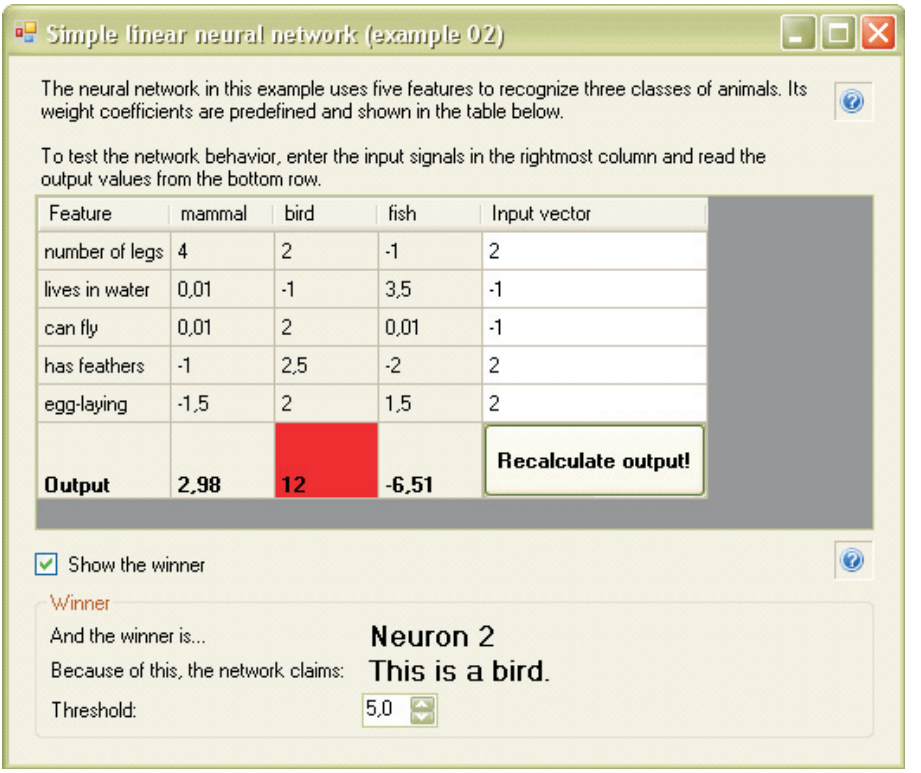
Natomiast sieć popadnie w rozterkę, gdy się jej pokaże węża (nie ma toto nóg, żyje na lądzie i rodzi się z jajek ???). Wtedy wszystkie neurony ze wstrętem oznajmiają, że coś takiego nie jest żadnym przyzwoitym zwierzęciem (wszystkie wyjścia mają wartości ujemne), co w kontekście przyjętych zasad klasyfikacji jest sensowne.

Opisana sieć jest bardzo prymitywna i popełnia niekiedy błędy, na przykład uparcie uważa żółwia za ssaka (ma 4 nogi, żyje na lądzie, ale przecież jest jajorodny, więc powinny być wątpliwości...), a nawet zalicza do ssaków ryby dwudyszne (takie żyjące podczas suszy na lądzie – proszę sprawdzić) – ale to już także taka swoista uroda sieci neuronowych – potrafią się one tak uroczo mylić... Jeśli ktoś z Was nigdy nie popełnił błędu – niech pierwszy rzuci w monitor kamieniem!

## 4.8. Jak i po co wprowadza się w sieci neuronowej rywalizację?

W praktycznych zastosowaniach korzysta się czasem z dodatkowego mechanizmu „rywalizacji" między neuronami, który w niektórych zastosowaniach pozwala uzyskiwać znacznie lepsze wyniki działania sieci. Zaobser-

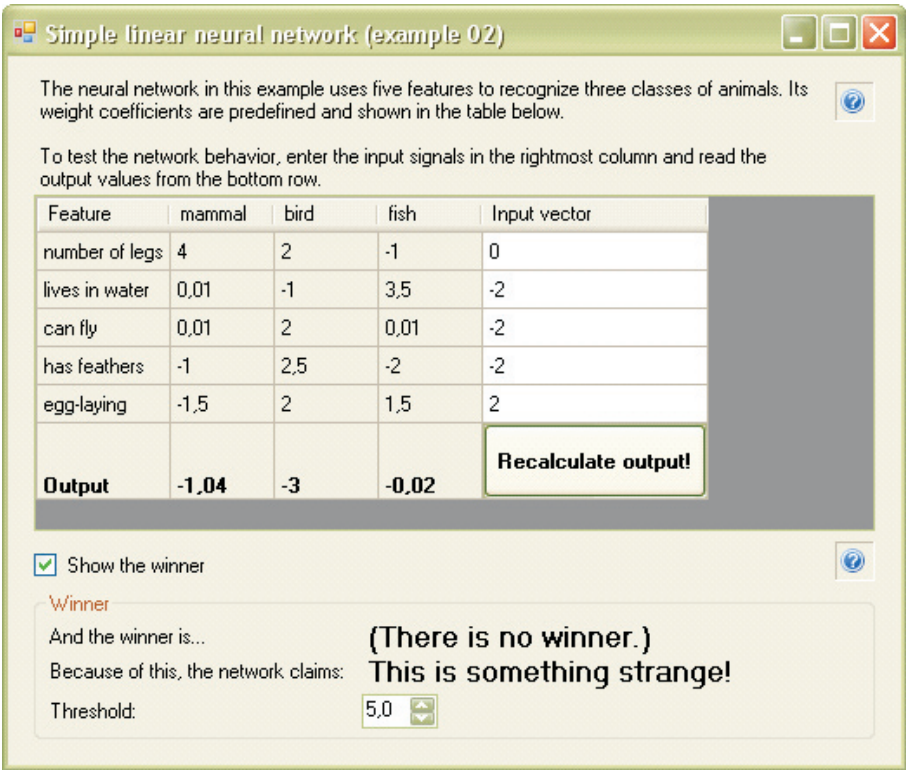
wowanie działania sieci z rywalizacją (*competition network*) możliwe jest po wprowadzeniu do dyskutowanej wyżej sieci rozpoznającej różne zwierzątka – elementu porównującego ze sobą sygnały wyjściowe wszystkich neuronów i typującego wśród nich „zwycięzcę". Zwycięzcą w tej konkurencji zostaje neuron o największej wartości sygnału wyjściowego. Z wytypowaniem „zwycięzcy" mogą wiązać się różne konsekwencje (na przykład tylko temu jednemu neuronowi można nadać prawo uczenia się – jak to ma miejsce w sieciach Kohonena, które opiszę Ci starannie, ale znacznie później), najczęściej jednak wytypowanie zwycięzcy służy do tego, by silniej spolaryzować wyjściowe sygnały z sieci – na przykład tylko neuron będący „zwycięzcą" ma prawo wysłać swój sygnał na zewnątrz, wszystkie pozostałe sygnały są natomiast zerowane. Taka zasada działania sieci, nazywana czasem **WTA** (*Winner Takes All* – zwycięzca zabiera wszystko) pozwala łatwiej interpretować zachowanie sieci (szczególnie wtedy, gdy ma ona wiele wyjść), ale niesie ze sobą pewne niebezpieczeństwa (wzmiankowane wyżej). Najlepiej sam się przekonaj, jak to działa, wprowadzając element rywalizacji do wcześniej wypróbowanego




Rys. 4.21. Przykład działania programu **Example 02** z włączoną rywalizacją neuronów

programu symulującego działanie sieci rozpoznającej zwierzęta. W tym celu uruchom jeszcze raz program **Example 02**; tym razem jednak zaznacz pole **Show the winner**. Spowoduje to, że program po „przepuszczeniu" danych wejściowych przez sieć zaznaczy na czerwono neuron, którego wyjście jest największe, oraz ogłosi jednoznaczny „werdykt". Przykład takiego działania pokazałem na rysunku 4.21.

Zauważ, że przy rywalizacji zakładamy, że tylko dodatnie wyjścia stanowią podstawę do podjęcia decyzji. Jeśli wszystkie sygnały wyjściowe będą mniejsze niż wartość oznaczona w programie jako „próg" (możesz ją ustawić dowolnie, wystarczy zmienić wartość w polu **Threshold**) – jako sygnał wyjściowy powinien zostać przyjęty sygnał **braku rozpoznania** (rys. 4.22).




Rys. 4.22. Sieć z rywalizacją usiłująca rozpoznać węża

Gorąco Cię zachęcam, abyś skorzystał z możliwości programu **Example 02** i spróbował wykonać kilka eksperymentów także i z opisaną wyżej siecią z rywalizacją. Przekonasz się, że sieć ta – w odróżnieniu od wcześniej zde-

finiowanej „zwykłej" sieci liniowej – ma kilka sympatycznych właściwości: podaje odpowiedzi znacznie bardziej kategoryczne, a w dodatku – tekstowo, a nie jako liczby wymagające dodatkowej interpretacji. Sieć ta ma jednak także poważne ograniczenia, które poznasz w toku lektury dalszych rozdziałów książki.

## 4.9. Jakie są dalsze możliwości wykorzystania sieci neuronowej?

Opisana wyżej sieć przeznaczona była do rozpoznawania pewnych zestawów informacji, traktowanych jako zbiór cech pewnych rozpoznawanych indywiduów. Nie jest to jednak jedyne możliwe zastosowanie opisywanych tu prostych jednowarstwowych sieci zbudowanych z neuronów o liniowej charakterystyce. Sieci tego typu bywają także dość chętnie wykorzystywane do wielu innych celów, na przykład do filtracji sygnałów (zwłaszcza jako filtry adaptacyjne, o zmieniających się właściwościach w zależności od potrzeb). Mają one także liczne zastosowania przy różnego rodzaju transformacjach sygnałów (mowy, muzyki, sygnału video, sygnałów medycznych – EKG, EEG, EMG) – na przykład mogą wydobywać widmo sygnału albo rozkładać wejściowe dane wg tzw. metody składowych kanonicznych PCA (*principal components analysis*). Wymieniam tu tylko kilka przykładowych zadań, ale oczywiście wykorzystanie każdej sieci – nawet takiej prostej, jak wyżej opisana – może być bardzo różne.

O wszystkim decyduje bowiem przyjęty zestaw wag dla wszystkich używanych neuronów. Dobierając inny zestaw wag, otrzymujemy inne działanie sieci, tak jak zmieniając program otrzymujemy inne działanie konwencjonalnego komputera. W opisywanych wyżej przykładach korzystaliśmy z wag narzucanych arbitralnie (trzeba było samemu wymyślić, jakie współczynniki wag mają mieć wszystkie używane neurony), co można interpretować w taki sposób, że narzucaliśmy sieci program działania. Dla niewielu neuronów występujących w sieci, a także dla prostej i w miarę oczywistej interpretacji współczynników wag w niej występujących (jak to na przykład było w przykładzie związanym z rozpoznawaniem zwierząt) – takie „ręczne programowanie" sieci może mieć miejsce i może dawać dobre wyniki. Jednak w praktycznych zastosowaniach występują sieci mające bardzo wiele elementów, w których rola i zadania pojedynczego neuronu są trudne do prześledzenia, dlatego bardziej użyteczne są oczywiście sieci, które same wybierają sobie współczynniki wag w wyniku procesu uczenia. W związku z tym w następnym rozdziale zajmiemy się starannie tym, co najważniejsze – zagadnieniami uczenia sieci.

## 4.10. Pytania, zadania i ćwiczenia do samodzielnego wykonania

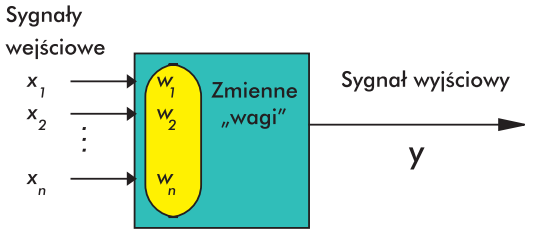
1. Jakie właściwości muszą mieć wagi neuronu oraz jego sygnały wejściowe, żeby neuron dawał:
   a. silny pozytywny sygnał wyjściowy?
   b. silny negatywny sygnał wyjściowy?
   c. sygnał wyjściowy o wartości zbliżonej do zera?
2. Jak sprawić, żeby neuron wyraźnie wyróżniał jedno ze swoich wejść (na przykład żeby kolor kwiatka był dla niego ważniejszy niż zapach)?
3. Jak można interpretować dodatnie i ujemne wagi przypisane do poszczególnych wejść neuronu?
4. Jak można interpretować dodatnie i ujemne sygnały pojawiające się na poszczególnych wejściach neuronu?
5. Czy neuron, na którego **wszystkich** wejściach pojawiły się wyłącznie ujemne sygnały, musi koniecznie dać na wyjściu sygnał ujemny?
6. Czy liczba sygnałów wejściowych do neuronu jest w jakiś sposób limitowana?
7. Sprawdź, czy sieć neuronowa modelowana przez program **Example 02** uzna **delfina** za ssaka czy za rybę?
8. Co się osiąga dzięki rywalizacji wprowadzonej do sieci neuronowej?
9. Zbadaj, do jakiej grupy zwierząt sieć modelowana przez program **Example 02** zaliczy **nietoperza.**
10. Spróbuj sprawdzić, czy sieć neuronowa modelowana przez program **Example 02** rozpozna, że **dinozaury nie** były ani ssakami, ani ptakami, ani rybami (były bowiem gadami, a takiej kategorii w sieci nie ma). Do jakich znanych sieci zwierząt dinozaury okażą się najbardziej podobne?
11. Czy sieć neuronowa z rywalizacją zawsze wyłania „zwycięzcę”? Czy to dobrze, czy źle?
12. **Zadanie dla zaawansowanych:** Rozbuduj sieć neuronową rozpoznającą zwierzęta, dodając dalsze rozpoznawane klasy zwierząt (na przykład zwierzęta drapieżne i żywiące się roślinami/wodorostami) oraz podając dodatkowe dane opisujące cechy zwierzęcia (na przykład ostrość zębów/dzioba oraz zdolność do szybkiego biegu/lotu/pływania).

# 5. Uczenie prostych liniowych sieci jednowarstwowych

## 5.1. Jak zbudować ciąg uczący?

W tym rozdziale musisz dowiedzieć się, jak się uczy najprostszą sieć. Ja wiem, była już o tym mowa, ale czym innym jest wiedza teoretyczna, wynikająca z przeczytania opisu algorytmu, a czym innym wiedza praktyczna, polegająca na zrobieniu pewnych rzeczy „*własnymi rękami*" – i zobaczeniu wyników. Pozostaniemy podczas opisywanych tu eksperymentów nadal przy sieciach **liniowych**, gdyż na neuronach liniowych wchodzących w skład właśnie takich sieci prowadziłeś badania w poprzednim rozdziale, a także zbudowałeś już pierwszą prostą sieć liniową i ją także trochę przebadałeś – i dlatego zapewne dobrze już wiesz, jak takie neurony oraz takie sieci się zachowują. W następnym rozdziale (numer 6) pokażę Ci także sieci **nieliniowe** i wtedy poznasz w praktyce, jak taka sieć działa i jakie są jej właściwości. Dlatego pozwól mi, że dopiero wtedy (na zasadzie odróżnienia) dokładniej wyjaśnię Ci, co oznacza przymiotnik **liniowy,** powtarzany z naciskiem przy nazwach sieci używanych w tym rozdziale. Na razie wystarczy, że uwierzysz mi na słowo, że sieci liniowe są znacznie prostsze od tych nieliniowych, łatwiej więc Ci będzie stawiać im zadania przy uczeniu i oceniać jego efekty. Zaczniemy ponadto zabawę od zadania najprostszego z prostych – od uczenia pojedynczego neuronu. Użyjesz do tego celu programu **Example 03**, pozwalającego na symulacyjne badanie i automatyczne trenowanie takiego pojedynczego liniowego neuronu (rys. 5.1).

Zanim jednak wprowadzisz do swojego komputera, wypróbujesz i (oczywiście!) ulepszysz ten mój program uczący – potrzebna jest pewna uwaga wprowadzająca. Otóż podczas eksperymentów polegających na badaniu działania pojedynczego neuronu i całej sieci (w poprzednim rozdziale) korzystałeś z możliwości **ręcznego** wprowadzania wszystkich potrzebnych sygnałów, co




Rys. 5.1. Struktura i podstawowe elementy podlegającego uczeniu sztucznego neuronu

dawało Tobie pełną kontrolę nad eksperymentem. Wtedy tak było łatwo, miło i wygodnie. Z chwilą jednak podjęcia prób **uczenia** trochę większej sieci trochę trudniejszych zadań – pieszczoty się kończą. Tu niekiedy potrzebne będą setki i tysiące eksperymentów, zanim coś sensownego zacznie się wyłaniać z początkowego chaosu. Oczywiście, można sobie wyobrazić masochistę, który kilka tysięcy razy wprowadza „z palca” te same dane w celu nauczenia sieci poprawnego rozwiązania jakiegoś zadania, ale zakładam, że tacy wariaci są jednak wśród Czytelników tej książki w zdecydowanej mniejszości.

Dlatego od początku założymy, że proces uczenia musi być oparty na zapamiętanym w komputerze, wcześniej przygotowanym tak zwanym **zbiorze uczącym,** w którym powinny być zawarte zarówno sygnały wejściowe (do wszystkich neuronów sieci), jak i wzorce poprawnych (wymaganych) sygnałów wyjściowych, z którymi algorytm uczący będzie konfrontował rzeczywiste zachowanie sieci. W moich programach format rekordów zbioru uczącego będzie następujący:

*komentarz (ułatwiający śledzenie, co się dzieje)*
*zestaw sygnałów wejściowych*
*wzorcowy zestaw sygnałów wyjściowych*

Zakładać przy tym będę, że w zestawie sygnałów wejściowych podawane będą pięcioelementowe wektory (porcje po pięć sygnałów dla pięciu wejść rozważanego neuronu), zaś wzorcowy sygnał wyjściowy będzie jeden (jako że masz do dyspozycji chwilowo tylko jeden neuron). Te parametry sieci umieszczone są w pierwszej linii pliku zawierającego ciąg uczący. Ciąg uczący może być dowolnie długi (prawdę mówiąc, im więcej przykładów prawidłowo rozwiązanych zadań zdołasz sieci pokazać – tym lepiej), więc plik zawierający podane wyżej zestawy informacji bywa niekiedy naprawdę duży. Ale na początek, dla programu uczącego jeden neuron, proponuję użyć takiego oto krótkiego pliku:

*5, 1*
*A typical object that should be accepted*
*3, 4, 3, 4, 5*
*1*

*A typical object that should be rejected*
*1, –2, 1, –2, –4*
*–1*
*An untypical object that should be accepted*
*4, 2, 5, 3, 2*
*0.8*
*An untypical object that should be rejected*
*0, –1, 0, –3, –3*
*–0.8*

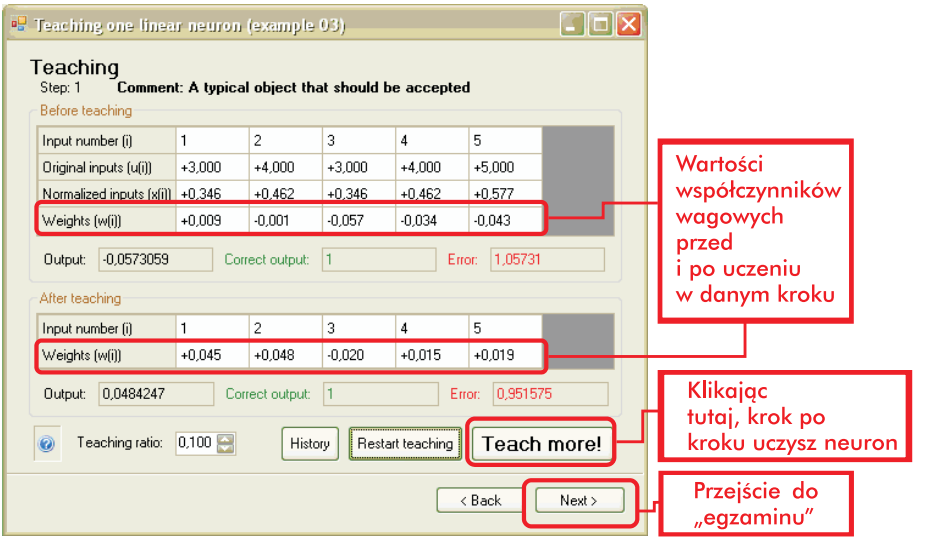
Podany wyżej tekst stanowiący plik uczący dla programu **Example 03** umieszczony jest w pliku o nazwie **Default teaching set 03.txt.** Program **Example 03** wie o tym, więc na początku zaproponuje Ci użycie właśnie tego pliku uczącego. Możesz jednak oczywiście zbudować swój własny, całkiem inny plik z danymi do uczenia sieci i podłączyć go do opisanego niżej programu. W ten sposób możesz nauczyć swój neuron rozpoznawania zupełnie innych wzorców albo możesz go zmusić, żeby zbudował **model** jakiegoś zjawiska (to znaczy, żeby aproksymował jakąś zależność między sygnałami wejściowymi a sygnałem wyjściowym – na przykład wynikającą z obserwacji medycznych albo z pomiarów fizycznych). Pamiętaj tylko, że jest to neuron **liniowy**, więc potrafi najwyżej nauczyć się przekształcać sygnały, podobnie jak robią to na przykład metody korelacji i wielowymiarowej regresji liniowej (to takie nazwy metod matematycznych używanych przez naukowców do statystycznego opisywania wyników badań eksperymentalnych), ale nie lepiej! Mądrzejsze neurony i bardziej uniwersalne sieci **nieliniowe** też poznasz, ale nie tak od razu...

Jednak proponuję, żebyś na początku skorzystał raczej z mojego pliku. Oczywiście, jak tylko wypróbujesz ten program i zobaczysz, że to wszystko dobrze działa – będziesz mógł do woli zabawiać się z własnymi danymi, ale wiedząc, na podstawie jakich danych uczysz swoją sieć – będę mógł opisać niżej, co podczas tej nauki widzisz i co wynika z tego, że widzisz właśnie to, a nie coś innego.

## 5.2. Jak można nauczać jeden neuron?

Zacznijmy od tego, że wczytasz program **Example 03** i spróbujesz go uruchomić. Zauważ, co robi opisany program: wczytuje dane z pliku (który – jak się umówiliśmy – nazywa się **Default teaching set 03.txt**) i najpierw sam próbuje prawidłowo zaklasyfikować obiekty, których dane (w postaci sygnałów wejściowych dla neuronu) są w pliku kolejno zapisane. Jeśli mu się nie uda – a neuron może się o tym sam przekonać, bo w pliku **Default teaching set 03.txt** znajdują

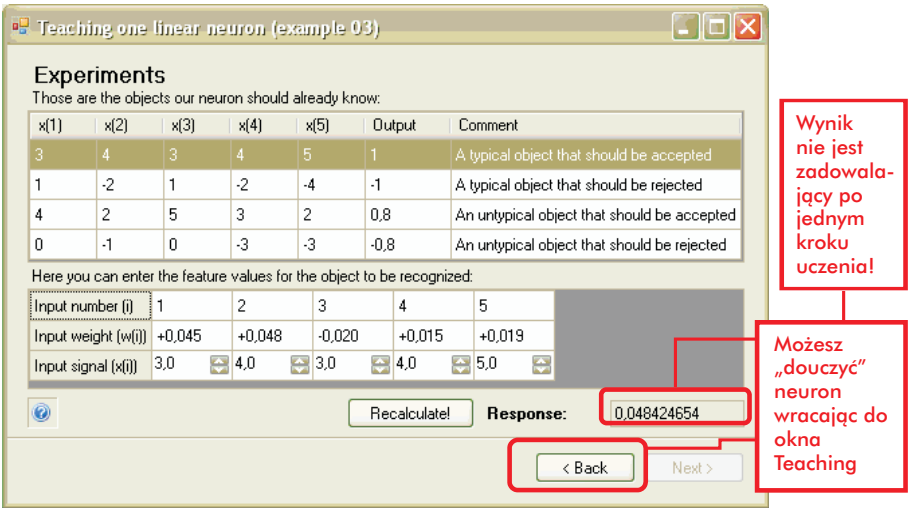
się także zapisane przez nauczyciela wzorce prawidłowych odpowiedzi – to program modyfikuje (zgodnie z algorytmem omówionym szczegółowo w jednym z wcześniejszych rozdziałów) współczynniki wagowe modelowanego neuronu. W ten sposób uczy go lepszego wykonywania postawionego zadania. Podczas symulowanego uczenia na ekranie swojego komputera możesz dokładnie śledzić w kolejnych krokach postęp uczenia, obserwując, jak zmieniają się współczynniki wag i błąd. Na początku błędy są – rzecz jasna – duże (rys. 5.2).




Rys 5.2. Początek procesu uczenia neuronu

Jeśli teraz poddasz sieć „egzaminowi” (do tej fazy możesz przejść w każdej chwili procesu uczenia, klikając przycisk **Next),** to sprawdzisz, co ona właściwie umie. Jednak na początku uczenia przekonasz się niestety, że Twoja nadzieja, iż neuron powinien już znać pokazywane mu obiekty, okaże się fatalnie nieprawdziwa (patrz rys. 5.3).

Żeby uzyskać sensowne działanie neuronu, trzeba go zatem trochę poduczyć. Klikając przycisk **Back** przejdź do poprzedniego okienka **Teaching** i cierpliwie ucz dalej neuron, klikając wytrwale, wiele razy, przycisk **Teach more!** Jeśli po pewnym czasie, obserwując zgłaszane przez program błędy, zauważysz, że błąd jest już w miarę niewielki, a jego zmiany (zmniejszanie) w następstwie kolejnych kroków procesu uczenia nie są już szczególnie znaczące (por. rys. 5.4) – to wtedy możesz przerwać proces uczenia i spróbować ocenić wiedzę  sieci za pomocą ponownego egzaminu. Wyniki powinny być lepsze – zarówno gdy pokażesz sieci któryś z obiektów ciągu uczącego (rys 5.5), jak i gdy pokażesz jej obiekt nie należący do ciągu uczącego (czyli



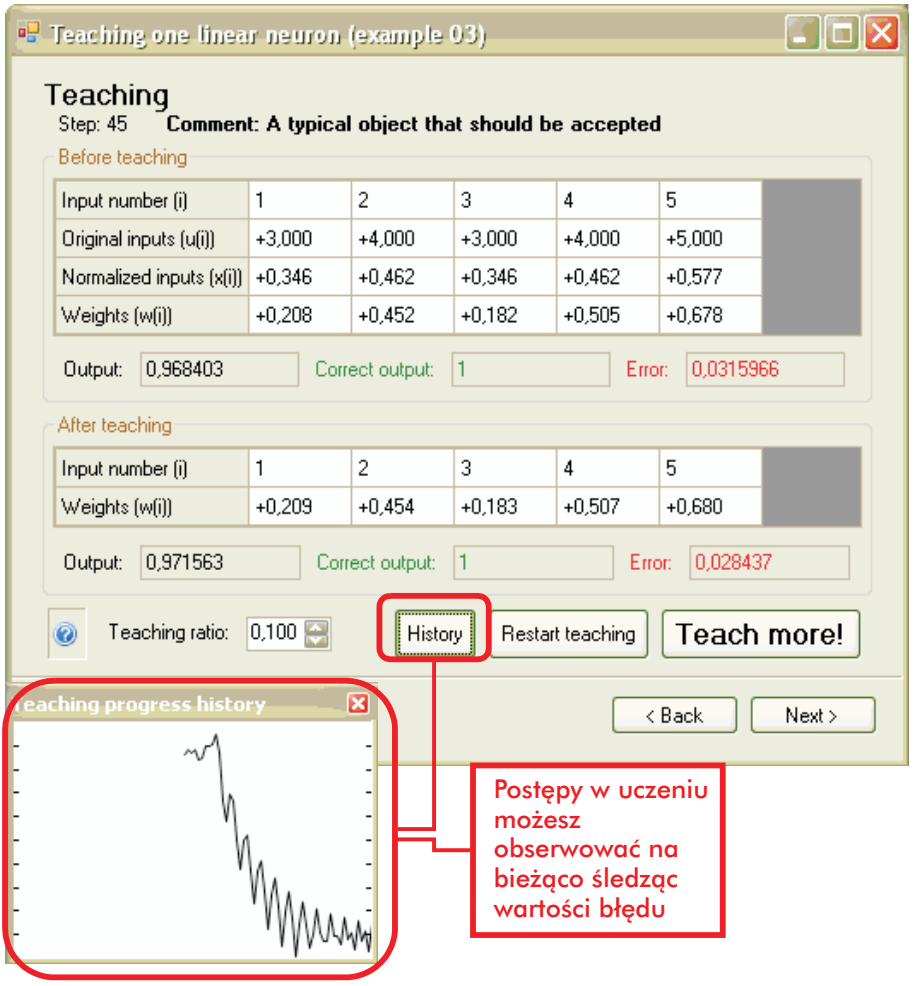

Rys. 5.3. Nieudany wynik egzaminu przy źle wytrenowanym neuronie (sieć bardzo słabo rozpoznaje pokazany obiekt)

zupełnie nowy, nie znany sieci uprzednio), ale **podobny** do obiektów ciągu uczącego (rys. 5.6). Podczas uczenia możesz skorzystać z obserwacji historii jego przebiegu, pokazującej się (na życzenie) w postaci wykresu zmian wartości błędu popełnianego przez sieć w kolejnych krokach procesu uczenia. Ten pożyteczny i pouczający wykres możesz przywołać w każdej chwili na swój ekran klikając przycisk **History**

Program pokazuje Ci zarówno błędy (jako główne elementy, mające wpływ na przebieg procesu uczenia), jak i wartości współczynników wag – przed korektą i po korekcie związanej z procesem uczenia. Pamiętając (z poprzednich rozdziałów), że dobrze nauczony neuron powinien wytworzyć sobie (w postaci odpowiednich wartości współczynników wagowych) „wzorzec" obiektu, który ma być rozpoznawany, możesz więc natychmiast ocenić, czy proces uczenia postępuje w dobrą stronę, czy też „idzie w krzaki" (oj, bywa, tak, bywa...).

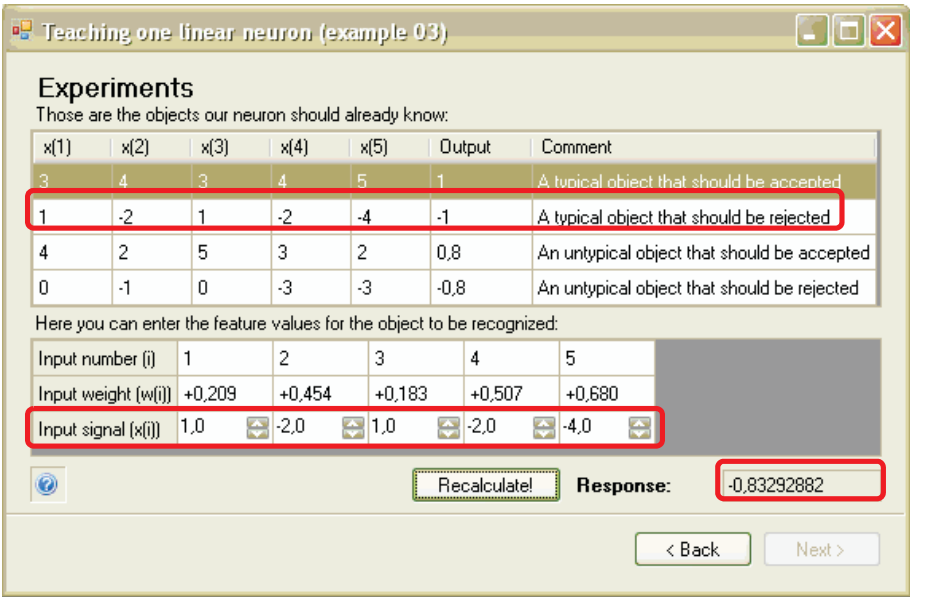
Zauważ, że przy niewielkiej liczbie elementów ciągu uczącego, które w dodatku pokazywane są nauczanemu neuronowi w kółko (bez korzystnego i zalecanego w jednym z wcześniejszych rozdziałów „tasowania" ich kolejności) zbyt wiele nie możesz wymagać – i zapewne w trudnych zadaniach zbyt wielkich sukcesów w procesie uczenia nie uzyskasz, ale dla prostych zadań (takich, jak zadanie umieszczone w pliku **Default teaching set 03.txt**) neuron uczy się zaskakująco szybko i skutecznie.

Zauważ też, jak wygląda dynamika procesu ucznia. Już pierwszych kilka pokazów daje znaczący postęp w zakresie polepszenia trafności działania

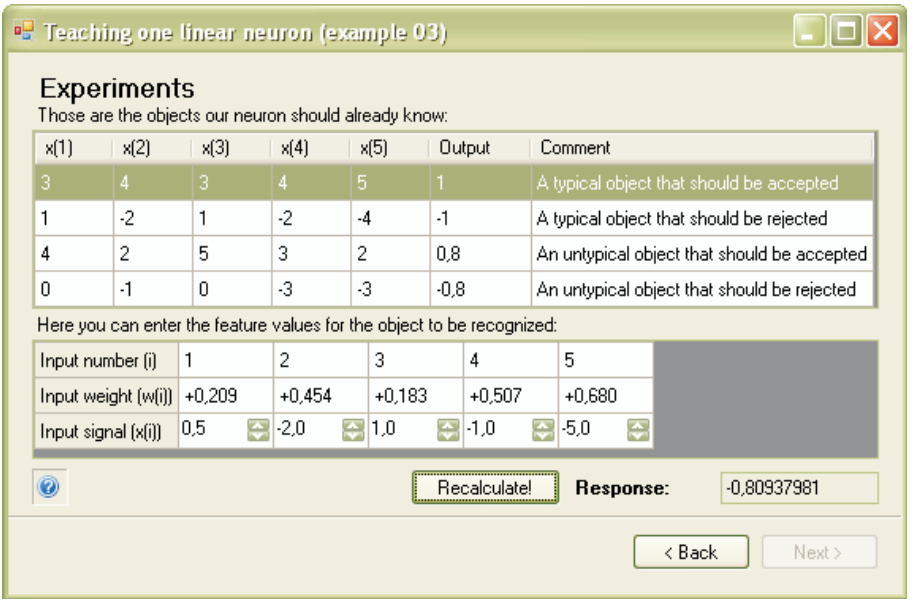



Rys. 5.4. Zaawansowany etap procesu uczenia charakteryzuje się małą wartością błędu w każdym kroku i niewielkim jego spadkiem po uczeniu

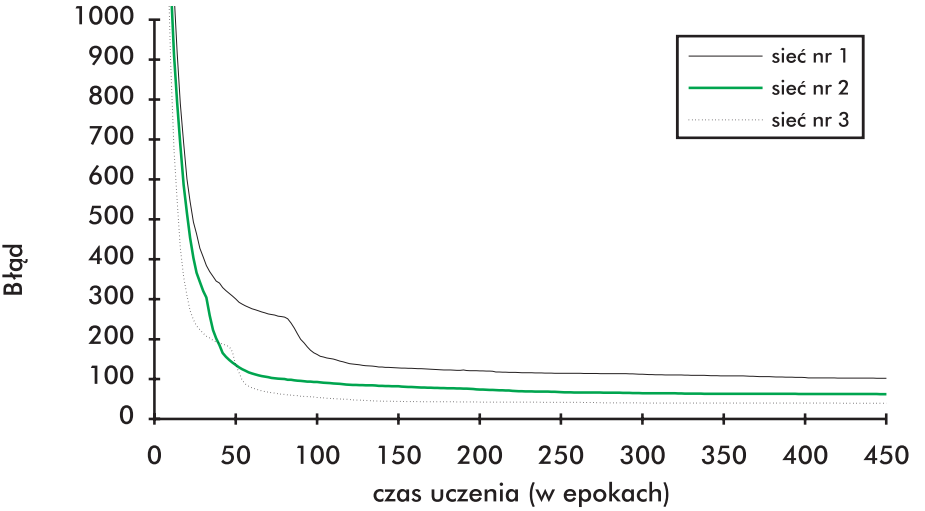
neuronu, natomiast potem szybkość uczenia spada, ale nadal daje się obserwować malejący błąd, jaki sieć popełnia w kolejnych krokach. Przypomnę Ci rysunek, ilustrujący typowy przebieg zmian wartości błędu w trakcie uczenia – pojedynczego neuronu, albo i całej sieci (rys. 5.7). Spróbuj sprawdzić, jak by wyglądał wykres przebiegu procesu uczenia w Twoich eksperymentach? Spróbuj także sprawdzić, że mimo stosowania zawsze tego samego ciągu uczącego – uzyskasz **różne** przebiegi procesu uczenia w kolejnych eksperymentach, ponieważ neuron będzie za każdym razem startował od innych (przypadkowo losowanych) początkowych wartości współczynników wag.




Rys. 5.5. W zaawansowanym okresie procesu uczenia sieć bez trudu zdaje egzamin polegający na odrzuceniu obiektu, który powinien być odrzucony




Rys. 5.6. W zaawansowanym okresie procesu uczenia sieć wykazuje zdolność do generalizacji, gdyż zdecydowanie odrzuca obiekt, który jest tylko **podobny** do obiektu ciągu uczącego, który powinien być odrzucony




Rys. 5.7. Malenie błędu w trakcie procesu uczenia dla różnych wartości początkowych wag w badanej sieci

Program **Example 03** jest dość „przyjazny" w użytkowaniu. Możesz z jego pomocą swobodnie eksperymentować z neuronem, na przykład przerywać proces uczenia i poddawać neuron „egzaminowi", badając, jak zachowa się po podaniu sygnałów próbnych – najlepiej podobnych do tych, których się uczył, ale nie identycznych. W każdej chwili możesz też wrócić do przerwanego procesu uczenia i trochę „podrasować" neuron, zanim poddasz go kolejnemu egzaminowi. W ten sposób możesz trochę poduczyć neuron, potem go przeegzaminować, znowu poduczyć, zastosować ponowny egzamin itd. Radzę Ci tylko stosować kolejno coraz dłuższe odcinki uczenia pomiędzy kolejnymi egzaminami, gdyż zmiany zachodzące w trakcie uczenia w miarę upływu kolejnych „epok" (kolejnych cykli pokazu wszystkich elementów ciągu uczącego) są coraz mniej zauważalne. Warto poświęcić chwilę na tego typu eksperymenty, gdyż dają bardzo konkretny i szczegółowy pogląd (tym lepiej, że na bardzo prostym i łatwym do prześledzenia przykładzie) na jedną z bardziej frapujących cech sieci neuronowych – zdolność do uczenia się oraz do uogólniania zdobytej wiedzy.

## 5.3. Czy neuron może mieć wrodzone zdolności?

Radzę Ci również sprawdzić, jak na efekty uczenia wpływają „wrodzone zdolności" neuronu. W tym celu powinieneś ponowić kilka razy proces uczenia, uruchamiając program od początku i obserwując zachodzące zmiany. Losowy proces nadawania wartości początkowych współczynników wagowych

spowoduje, że będziesz obserwował różne tempo uczenia neuronu dla tych samych danych wejściowych – czasem neuron uczy się wręcz błyskawicznie (taki jest od urodzenia uzdolniony!), a czasem wymaga bardzo długiego czasu nauki, w trakcie której zdarzają się nawet okresy pogarszania wyników (błąd rośnie mimo intensywnego uczenia), jeśli neuron musi przezwyciężać w trakcie nauki pewne „wrodzone preferencje". Ten efekt przypadkowości przebiegu i wyników procesu uczenia bywa zaskakująco duży, co trzeba samemu zobaczyć, żeby w to uwierzyć. Dlatego nie oszczędzaj czasu i przeprowadź kilka doświadczeń z zaproponowanym programem, zmieniając w tekście programu zakres, w jakim ustalane są początkowe losowe wartości współczynników wag[1]. Możesz tego dokonać podając inne niż przyjęte domyślnie parametry inicjalizujące wagi początkowe neuronu w metodzie `InitializeTeaching() klasy ProgramLogic`. Na przykład możesz zamiast instrukcji

```
_examinedNeuron.Randomize(_randomGenerator, -0.1, 0.1);
```

zastosować instrukcję

```
_examinedNeuron.Randomize(_randomGenerator, -0.4, 0.4);
```

w wyniku czego początkowe wartości wag będą miały znacznie większy zakres zmian – a przez to także znacznie większy wpływ na przebieg uczenia sieci i na jego wyniki.

Uruchamiając kilka razy proces uczenia z tym samym ciągiem uczącym, łatwo zaobserwujesz, jak duży może być wpływ przypadkowo rozłożonych początkowych wartości wag. Neuron za każdym razem będzie się uczył zupełnie inaczej! W ten sposób poznasz namacalnie jeszcze jedną, demonizowaną niekiedy (*„wolna wola automatu?!"*) cechę sieci neuronowych – ich **indeterminizm,** czyli nieprzewidywalność zarówno przebiegu procesu uczenia, jak i jego wyników.

Weź przy tym pod uwagę następującą okoliczność: Sieć, której używasz do ćwiczeń opisywanych w tym rozdziale, jest raczej niewielka i mało skomplikowana, a samo zadanie, które sieć ma zrealizować, jest raczej łatwe. Natomiast w dużej sieci i przy rozwiązywaniu naprawdę skomplikowanego zadania sumowanie się wielu losowych efektów, takich jak przebadane w opisanym programie, może doprowadzić do całkiem nieprzewidywalnych zachowań sieci, co niekiedy wprawia w osłupienie „uczonych w piśmie", przyzwy-

[1] Niestety, to doświadczenie wymaga już zajrzenia do tekstu programu i zmiany jednej z jego instrukcji, a potem powtórnej jego kompilacji – co oznacza, że ta zabawa przeznaczona jest dla bardziej zaawansowanych Czytelników tej książki. Ale mam nadzieję, że stosując się do wyżej podanych wskazówek poradzisz sobie z tą sprawą.

czajonych do całkowitej i niezawodnej powtarzalności zachowań regularnych algorytmów numerycznych, wykorzystywanych w typowych programach komputerowych.

## 5.4. Jak mocno należy neuron uczyć?

Zaproponowany wyżej prosty programik demonstracyjny może być „poligonem doświadczalnym", na którym powinieneś zbadać jeszcze jeden ważny czynnik, wpływający na przebieg procesu uczenia, a mianowicie wpływ wartości współczynnika określającego szybkość uczenia na jego przebieg. Współczynnik ten można zmieniać w polu **Teaching ratio** w oknie **Teaching** programu **Example 03**. Dając większą wartość współczynnika uczenia, na przykład stosując

Teaching ratio = 0,3,

zamiast użytego w programie

Teaching ratio = 0,1,

możesz uzyskać szybszy efekt procesu uczenia, ale przebieg uczenia będzie wtedy bardziej „nerwowy" (z gwałtownymi zmianami wartości wag i z nagłymi skokami (w górę i w dół) wartości popełnianych przez sieci błędów. Możesz wypróbować kilka różnych wartości i dokładnie zaobserwować ich wpływ na proces uczenia. Musisz jednak pamiętać, że jeśli dasz za dużą wartość tego współczynnika – proces uczenia zacznie być chaotyczny i nie przyniesie żadnych pozytywnych rezultatów, ponieważ neuron będzie się „szamotać" od jednej skrajności do drugiej, a efekty końcowe będą opłakane – zamiast polepszać swoje wyniki neuron będzie odnotowywał coraz większe błędy. To także naprawdę warto zobaczyć na własne oczy!

Z kolei zbyt mała wartość współczynnika uczenia sprawi, że proces uczenia będzie postępował bardzo wolno, w praktyce postęp uczenia może być wręcz niezauważalny i jest wielce prawdopodobne, że użytkownik, stosujący sieć neuronową w praktyce, zniechęci się, nie widząc postępów w jej działaniu, i porzuci tę technikę, szukając bardziej efektywnych algorytmów.

Opisanym doświadczeniom warto nadać interpretację, nawiązującą do sytuacji, jakie mają miejsce w rzeczywistym procesie uczenia, w którym zamiast sztucznych sieci neuronowych uczestniczą rzeczywiste mózgi zdobywających wiedzę studentów. Łatwo się zorientować, że wartość współczynnika uczenia wyraża stopień „surowości" nauczyciela. Małe wartości tego współczynnika odpowiadają sytuacji, kiedy nauczyciel jest łagodny i wyrozumiały – dostrzega wprawdzie błędy ucznia i koryguje je, ale nie wymusza przemocą poprawnych odpowiedzi. Jak widziałeś na podstawie przeprowadzonych eksperymentów – taka pobłażliwość może prowadzić do słabych wyników.

Jednak za duże wartości współczynnika uczenia, czyli nadmierna surowość nauczyciela – też bywa szkodliwa. Sypiące się na ucznia surowe kary, zdecydowana i twarda nagana po każdym błędzie – prowadzą do frustracji, której wyrazem jest „miotanie się" neuronu od jednej skrajności do drugiej – bez rzeczywistego postępu procesu uczenia.

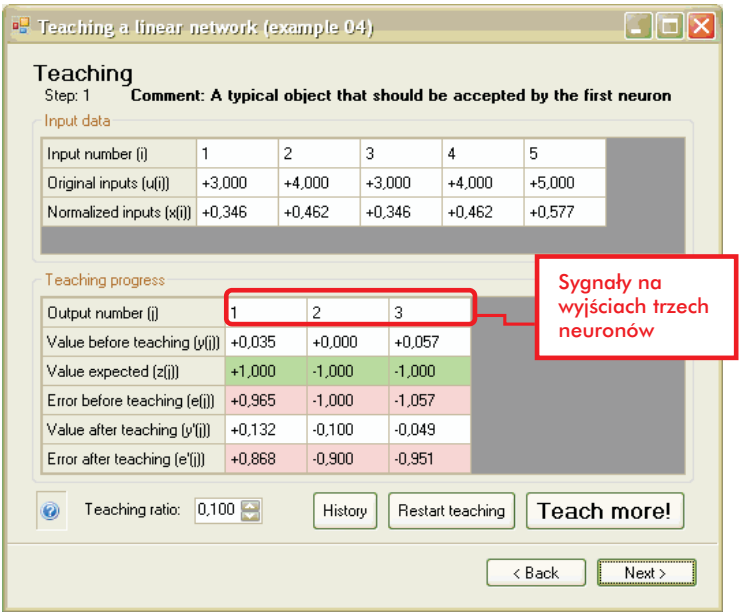
## 5.5. Jak uczyć prostą sieć?

Naturalną koleją rzeczy przejdziemy teraz od nauki pojedynczego neuronu do nauki całej sieci. Program **Example 04**, który w tym celu zbudowałem, jest bardzo podobny do opisanego wyżej programu dla jednego neuronu, będzie Ci więc łatwo go używać. Do eksperymentów z podanym wyżej programem trenującym sieć potrzebujesz też pliku ze zbiorem uczącym. Ja przygotowałem i wykorzystałem taki plik, nazywając go także **Default teaching set 04.txt**, ale umieszczając go w taki sposób, że jest on powiązany tym razem z programem **Example 04.** Zawartość tego **zbioru uczącego** jest taka, jak przytoczono niżej:

*5, 3*
*A typical object that should be accepted by the first neuron*
*3, 4, 3, 4, 5*
*1, –1, –1*
*A typical object that should be accepted by the second neuron*
*1, –2, 1, –2, –4*
*–1, 1, –1*
*A typical object that should be accepted by the third neuron*
*–3, 2, –5, 3, 1*
*–1, –1, 1*
*An untypical object that should be accepted by the first neuron*
*4, 2, 5, 3, 2*
*0.8, –1, –1*
*An untypical object that should be accepted by the second neuron*
*0, –1, 0, –3, –3*
*–1, 0.8, –1*
*An untypical object that should be accepted by the third neuron*
*–5, 1, –1, 4, 2*
*–1, –1, 0.8*
*An untypical object that should be rejected by all neurons*
*–1, –1, –1, –1, –1*
*–1, –1, –1*

Program, który teraz będziesz badał, informuje o stanie procesu uczenia i wartościach poszczególnych zmiennych nieco mniej szczegółowo niż

program wcześniej omówiony (**Example 03**), ponieważ przy większej liczbie neuronów zbyt dokładny wgląd w to, co robi każdy pojedynczy neuron, mógłby okazać się kłopotliwy – program zasypałby Cię lawiną informacji, wśród których trudno byłoby Ci wyłuskać te istotne. Dlatego wgląd w proces uczenia dawany przez program **Example 04** jest bardziej syntetyczny – od razu dla wszystkich trzech neuronów rozważanej sieci, jak widać m.in. na rysunku 5.8.

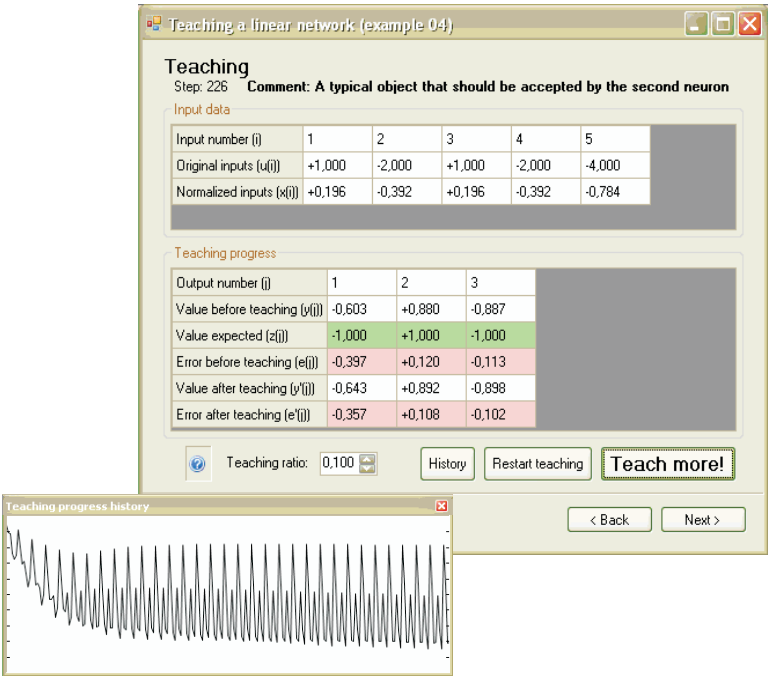



Rys. 5.8. Obraz stanu sieci na początku procesu uczenia w programie **Example 04**

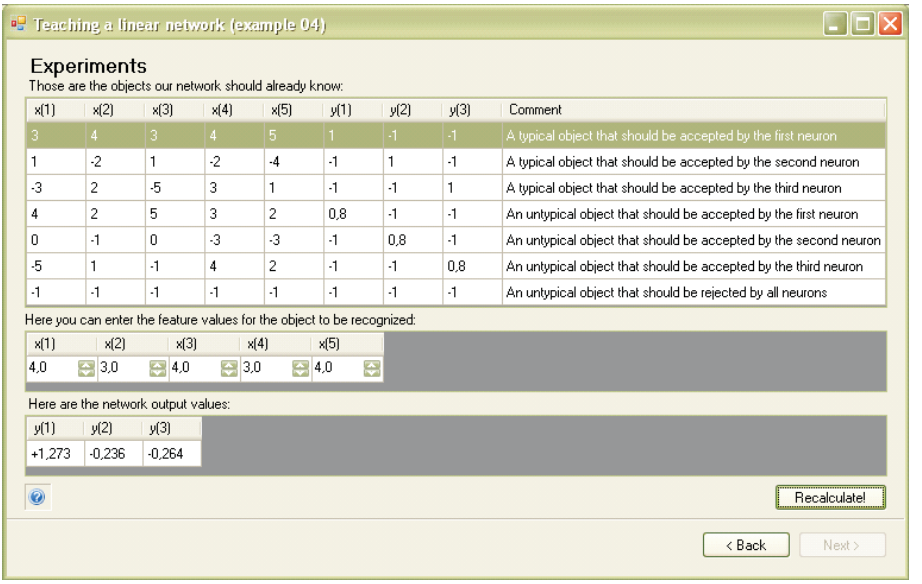
Nie przeszkadza to jednak w uzyskaniu wglądu w to, co najbardziej istotne – w postęp procesu uczenia, co można zobaczyć porównując rysunek 5.8 z rysunkiem 5.9.

Opisywany program, z tych samych powodów, które wymieniono wyżej, także w sposób nieco bardziej syntetyczny pokazuje zachowanie sieci w trakcie egzaminu (rys. 5.10), co jednak nie przeszkadza w sprawnym zorientowaniu się, czy sieć działa prawidłowo, czy nie.

Jak możesz zauważyć na rysunku 5.10 – sieć po nauczeniu daje sobie nie najgorzej radę nawet z takim obiektem, który nie był pokazywany w ciągu uczącym, ale jest podobny do obiektu, który miał pozytywnie rozpoznawać konkretny (w tym przypadku pierwszy) neuron. Zauważ, że podczas egza-




Rys. 5.9. Obraz stanu sieci na końcu procesu uczenia w programie **Example 04**

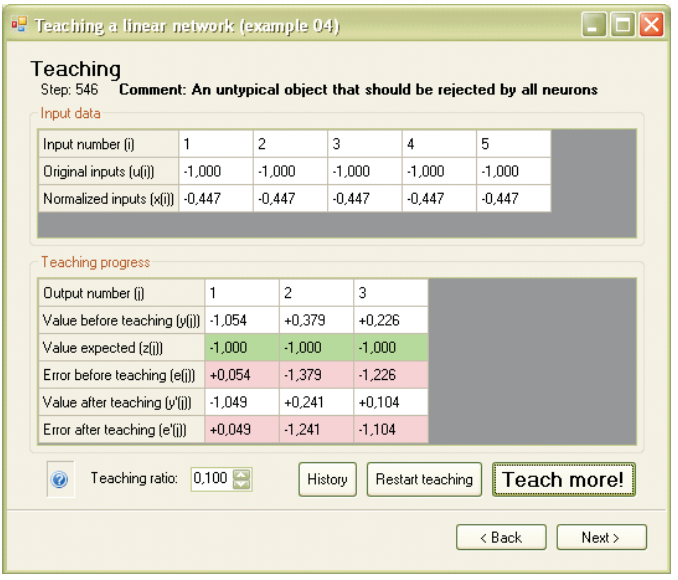



Rys. 5.10. Przebieg egzaminu sieci w programie **Example 04**

minu **tylko** pierwszy neuron ma dodatnią wartość wyjściowego sygnału, co zdecydowanie i jednoznacznie wskazuje, że rozpoznawanie zakończyło się sukcesem. Jednak nietypowość sytuacji spowodowała, że neurony nr 2 i 3, które powinny się były zdecydowanie wyrzec tego obiektu, miały wątpliwości – ich odpowiedzi były chwiejne i niezdecydowane.

Jest to sytuacja dość typowa. Wymagając od sieci **uogólnienia** zdobytej w trakcie uczenia wiedzy, prawie zawsze spotykamy się z jej niepewnym zachowaniem, polegającym na tym, że łatwiej jest osiągnąć sukces (na przykład poprawne pozytywne rozpoznanie), niż uzyskać dużą **pewność** i **niezawodność** działania sieci w odniesieniu do przypadków, które powinna twardo i zdecydowanie odrzucać.

Obserwując proces uczenia sieci, zauważysz zapewne, że wśród żądań, które jej stawiasz w trakcie uczenia (w postaci odpowiednich przykładów wchodzących w skład ciągu uczącego), są takie, które sieć potrafi rozwiązać łatwo i szybko, oraz takie, które przychodzą jej z dużym trudem. W odniesieniu do pokazanego przeze mnie wyżej przykładowego pliku **Default teaching set 04.txt** łatwe są wszystkie zadania polegające na konieczności pozytywnego rozpoznawania przez konkretne neurony konkretnych – typowych lub nietypowych – wzorców. Natomiast trudny do nauczenia okazuje się przykład obiektu, który wszystkie neurony powinny odrzucić. Na rysunku 5.11 pokazałem jeden z bardzo dalekich etapów procesu uczenia, kiedy po-




Rys. 5.11. Uczenie sieci w przypadku rozpoznawania kłopotliwych przykładów

zytywne rozpoznawanie konkretnych obiektów odbywa się już praktycznie bezbłędnie, a obiekt, który powinien być odrzucany – jak widać z rysunku – wciąż sprawia duże kłopoty.

Mało tego – można zauważyć, że kolejne nieudane próby dostosowania sieci do rozwiązywania także tego właśnie trudnego zadania prowadzą do **psucia** już stosunkowo dobrego stanu wyuczenia sieci w zakresie rozwiązywania podstawowych zadań (tzn. rozpoznawania typowych obiektów).

Jeśli napotkasz taką sytuację w praktyce – zamiast miotać się i długo, bezskutecznie uczyć sieć – zastanów się, czy nie da się tak przeformułować rozwiązywanego zadania, żeby opisana sytuacja wcale nie występowała. Zazwyczaj usunięcie z ciągu uczącego jednego czy kilku kłopotliwych przykładów radykalnie polepsza i przyspiesza proces uczenia. Jak bardzo poprawia to sprawność uczenia, możesz się sam przekonać, usuwając odpowiedni fragment z pliku **Default teaching set 04.txt** i ucząc sieć ponownie – ale już bez tego kłopotliwego elementu. Z pewnością zauważysz korzystną zmianę zarówno szybkości, jak i skuteczności uczenia.

## 5.6. Jakie są możliwości zastosowania takich prostych sieci neuronowych?

Nie zdziwię Cię zapewne (i mam nadzieję, że Cię również nie zasmucę), jeśli wyznam Ci, że sieć, którą przed chwilą badałeś, nie jest największym i najbardziej skomplikowanym neuro-komputerem, jaki zbudowano na świecie. Prawdę powiedziawszy, to była ona taka prosta, że niektórym z Czytelników mogła się wręcz wydawać prymitywna. A jednak nawet ta prosta sieć wykazywała całkiem ciekawe formy zachowania i potrafiła sobie całkiem zgrabnie radzić z dość skomplikowanymi zadaniami – bo kolektywne działanie wielu neuronów otwiera naprawdę szeroki zakres bardzo zróżnicowanych możliwości.

Spróbujmy pozwolić sobie w tym miejscu na jedną uwagę nieco bardziej ogólną. Otóż łatwo jest zauważyć, że między możliwościami pojedynczego neuronu a możliwościami całej sieci są pewne podobieństwa. W związku z tym sieć można rozpatrywać jako zbiorowość współpracujących neuronów, a wiedza o zachowaniu pojedynczych neuronów w dosyć naturalny sposób może być przeniesiona na całą sieć. Są jednak pewne możliwości unikatowo związane z sieciami, których pojedynczy neuron nie dostarczał. Na przykład sieci, nawet takie proste, jak te tutaj omawiane, pozwalają śmiało wkraczać w obszar rozwiązywania problemów **wielowymiarowych,** które innym metodom matematycznym i obliczeniowym mogą sprawiać nierzadko bardzo poważne trudności. Możesz się o tym sam przekonać, bo powiększywszy od-

powiednio rozmiar sieci przez przyjęcie większych wartości dla liczby wejść i dla liczby wyjść, możesz spróbować zastosować sieć do jakichś zadań praktycznych. W szczególności sieć może zostać użyta do modelowania różnych złożonych systemów, w których wiele przyczyn (sygnałów wejściowych) składa się na występowanie wielu skutków (sygnałów wyjściowych). O tym, jak wiele jest takich zastosowań, możesz się przekonać, wpisując na przykład do przeglądarki Google hasło „*Neural Networks Modeling*". Ja przed chwilą dostałem informację, że takich opisanych przykładów można w Internecie znaleźć blisko 30 milionów… Trudno byłoby znaleźć aż tyle praktycznych zastosowań dla działania pojedynczego neuronu!

Poza tworzeniem neuronowych modeli różnych złożonych systemów możemy takich właśnie (jak omawiane tutaj) prostych liniowych sieci neuronowych używać do adaptacyjnego przetwarzania sygnałów. To znowu ogromna i bardzo popularna dziedzina – nie chcę Ci nawet mówić, ile artykułów zasygnalizowała przeglądarka Google gdy jej podałem hasło *„Signal Processing*". Wynika to z faktu, że technika cyfrowa otworzyła możliwości komputerowej (a więc także neuronowej) obróbki bardzo wielu różnych sygnałów – na przykład mowy przekazywanej przez telefon, obrazów rejestrowanych przez aparaty fotograficzne albo kamery wideo, sygnałów opisujących stan organizmu pacjenta, dostarczanych przez nowoczesną aparaturę medyczną, wyników eksperymentów naukowych, też z reguły produkujących ogromne ilości sygnałów, zapisów z aparatury pomiarowo-kontrolnej w systemach automatyki przemysłowej – i niezliczonych dalszych.

Myślę, że dobrze będzie, abyś się chociaż odrobinę przyjrzał tej obszernej, ważnej i ciekawej dziedzinie, bo właśnie filtracja sygnałów (czyli usuwanie z tych sygnałów zakłóceń, które utrudniają ich odbiór, analizę i interpretację) stwarza dla sieci neuronowych szczególnie atrakcyjne pole zastosowań.

## 5.7. Czy sieć może się nauczyć filtrować sygnały?

Wyobraź sobie, że masz jakiś sygnał z nałożonym nań szumem. Inżynierowie od telekomunikacji, automatyki, elektroniki i mechatroniki męczą się z takimi sygnałami każdego dnia, więc na pozór niby nic w tym dziwnego. Jeśli spytasz takiego specjalistę, co zrobić, żeby oczyścić sygnał od tego szumu, to z poczuciem wyższości pouczy Cię, że należy zastosować filtr. Drążąc temat nieco dokładniej, dowiesz się, że filtr to takie urządzenie (najczęściej – elektroniczne), które użyteczny sygnał przepuszcza, a szum – zatrzymuje. Działa to bardzo dobrze i dlatego mamy sprawnie działające telefony, radioodbiorniki, telewizory itd. Jednak każdy specjalista potwierdzi, że dobry filtr da się zbudować tylko wtedy, gdy w szumie da się wyróżnić jakąś cechę, której nie

ma w użytecznym sygnale. Jeśli coś ma tę cechę – to jest przez filtr zatrzymywane jako szum. Jeśli nie ma – to jest przepuszczane. Proste i skuteczne!

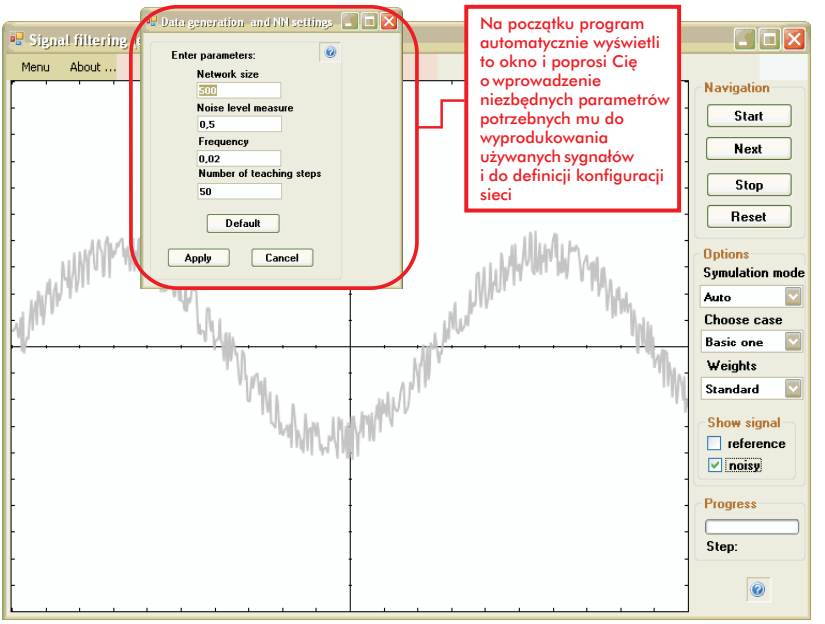
No tak, ale żeby to działało, to musisz sporo wiedzieć o szumie, który zakłóca Twój sygnał. Jak nie wiesz, to nie zbudujesz filtru, bo niby na jakiej podstawie miałby on (ten filtr) wiedzieć, co przepuszczać, a co zatrzymywać?

Niestety, często bywa tak, że nie wiesz, skąd się bierze szum zakłócający Twój sygnał – ani jakie są jego właściwości. Jeśli wysyłasz gdzieś daleko w Kosmos sondę, która ma Ci przysłać sygnały opisujące na przykład nieznaną planetoidę – to nie wiesz, jakie paskudztwa przyplączą się do sygnałów sondy podczas ich wędrówki przez miliony kilometrów przestrzeni międzyplanetarnej. Jak wtedy zbudować filtr?!

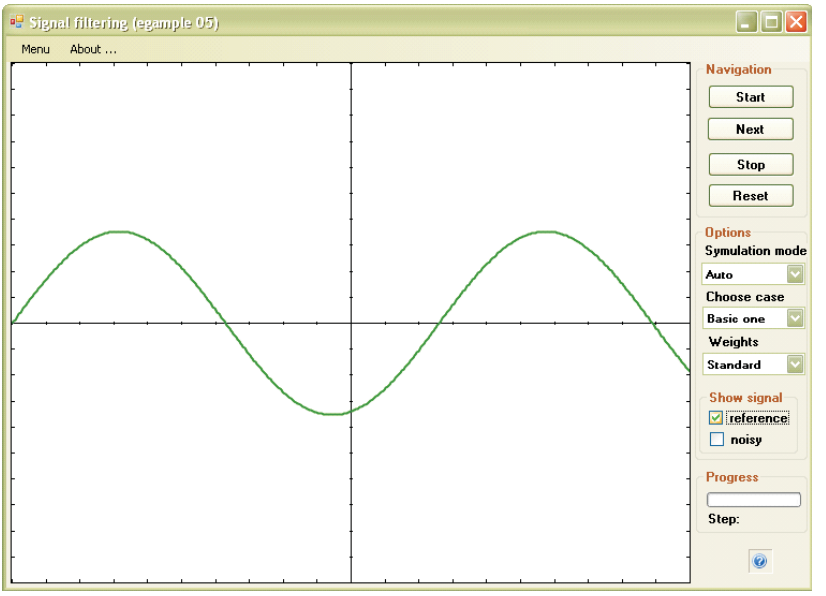
Otóż da się oddzielać pożyteczny sygnał od **nieznanych** zakłóceń stosując filtrację adaptacyjną. Adaptacja polega w tym przypadku na uczeniu urządzenia odbierającego sygnały nieznanych wcześniej zasad oddzielenia sygnału od szumu. W szczególności sieć neuronową można wytrenować tak, by filtrowała sygnał odsiewając szum. W tym celu jako sygnały wejściowe do sieci traktowane są próbki sygnału zakłóconego, a jako sygnały wyjściowe używane są próbki sygnału „czystego". Po pewnym czasie sieć nauczy się wydobywać niezakłócone sygnały wyjściowe z sygnału zaszumionego i będzie mogła znaleźć zastosowanie jako filtr.

Rozważmy konkretny przykład. Niech będzie dany sygnał wzorcowy – na przykład fragment sinusoidy – zarówno „czysty", jak i zaszumiony. Sygnały takie – w postaci pliku pozwalającego na uczenie sieci – mogą być wyprodukowane przez program **Example 05**. Program ten automatycznie wyprodukuje plik z danymi o nazwie **teaching_set**, który możesz użyć do treningu prostej sieci. Okienko, w którym można modyfikować parametry wymagane do wygenerowania tego pliku, pojawi się zaraz po uruchomieniu programu **Example 05.** Okienko to widzisz na rysunku 5.12. Niestety, w okienku tym program wymaga od Ciebie, żebyś podał rozmiar sieci (**Networks size),** spodziewaną wartość szumu **(Noise level measure),** częstotliwość **(Frequency)** oraz liczbę kroków procesu uczenia **(Number of teaching steps).** Patrząc na to wszystko, możesz się zdecydowanie zniechęcić do zabawy z tym programem, bo wygląda to tajemniczo i niezrozumiale, a w dodatku skąd byś Ty miał znać te wszystkie potrzebne wartości?

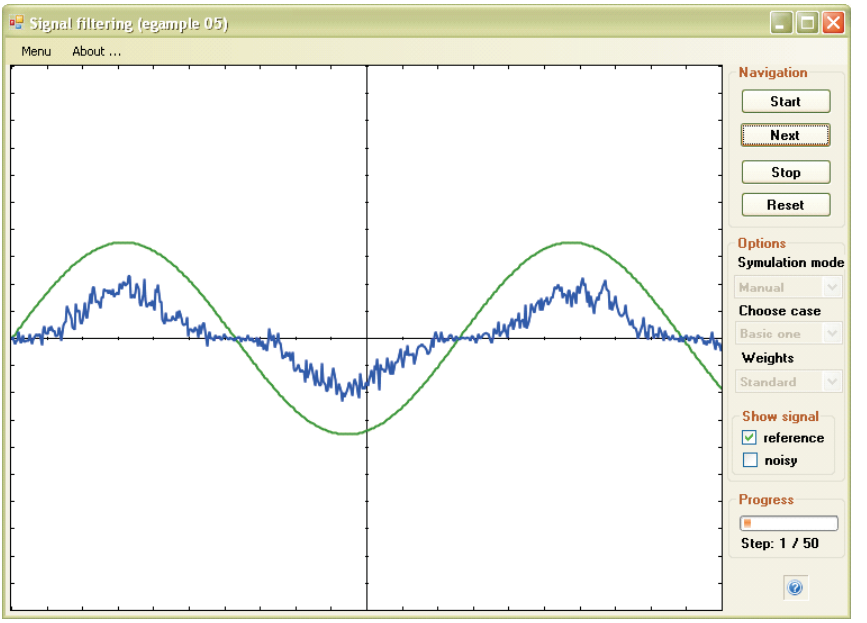
Nie jest jednak tak źle. Jak się dobrze przyjrzysz opisywanemu okienku, to zauważysz, że we wszystkich okienkach coś już jest wpisane. Są to wartości tak dobrane i wypróbowane przeze mnie, żeby program dobrze działał i pokazywał ciekawe efekty. Jeśli zatem nie masz sam lepszych pomysłów, to na początku możesz skorzystać z tych ustawień domyślnych i po prostu (nie robiąc nic więcej) zaakceptować je klikając przycisk **Apply.** Oczywiście po-



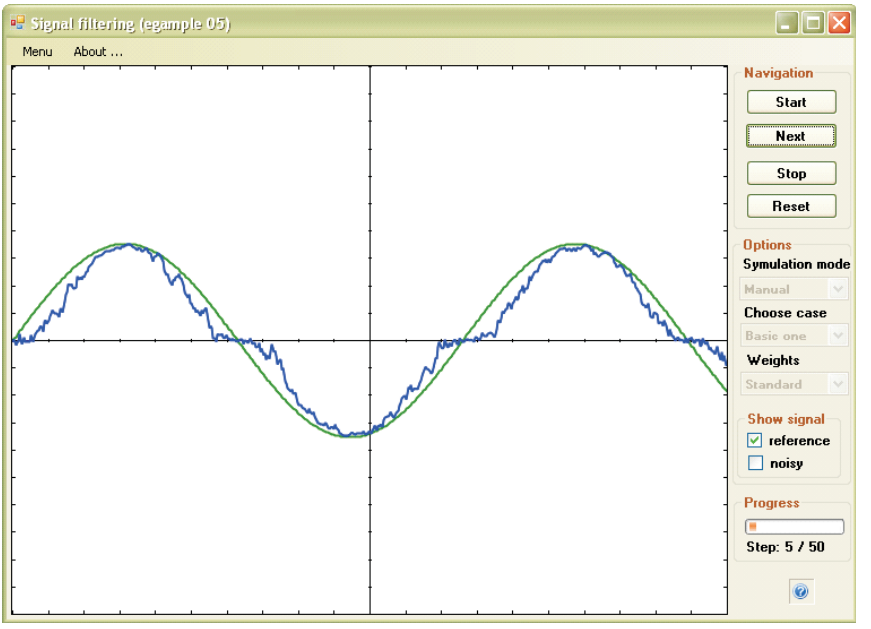

Rys. 5.12. Zakłócony sygnał, który ma filtrować sieć neuronowa



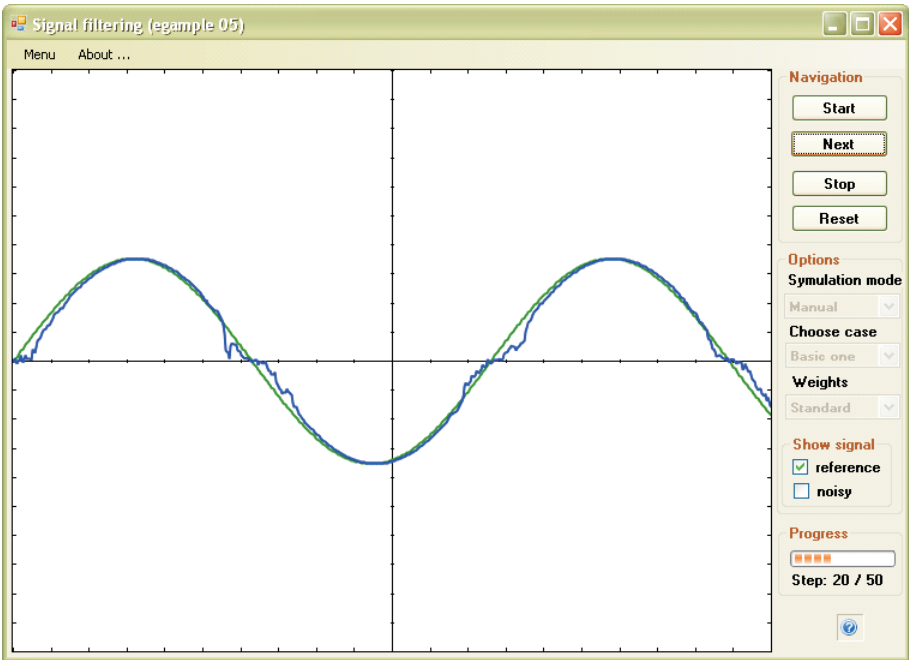

Rys. 5.13. Sygnał wzorcowy, który ma odtworzyć sieć neuronowa




Rys. 5.14. Wynik filtracji sygnału po jednym kroku uczenia




Rys. 5.15. Wynik filtracji sygnału po pięciu krokach uczenia




Rys. 5.16. Wynik filtracji sygnału po dwudziestu krokach uczenia

tem, jak zechcesz i jak potrafisz, to każdą z tych wartości będziesz sobie mógł dowolnie zmienić – chociażby po to, żeby zobaczyć „co by było, gdyby".

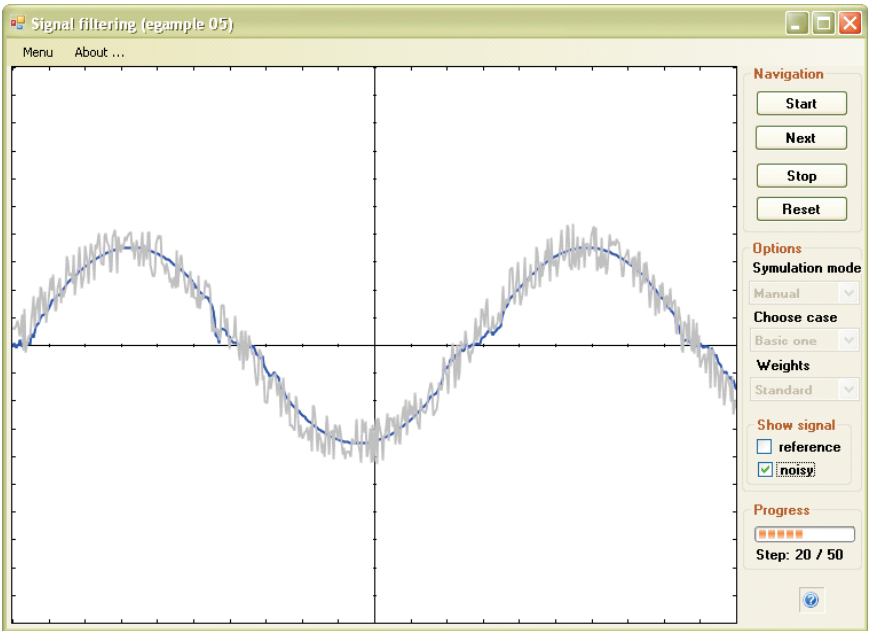
Swoboda, jaką daje Ci w tym zakresie rozważany program, jest bardzo duża. Wprowadzone już wartości możesz w każdej chwili skasować (przycisk **Cancel)**, a także możesz natychmiast odtworzyć ponownie cały zestaw zaprojektowanych przeze mnie parametrów klikając przycisk **Default.**

Jak z tego wynika, mimo groźnego wyglądu program **Example 05** naprawdę da się lubić, a korzystając z jego pracy będziesz mógł obserwować działanie sieci, która sama nauczy się filtracji sygnału. W trakcie eksperymentów sieć będzie poprawiała sygnał z kroku na krok, ale w dowolnej chwili możesz zobaczyć, jak wygląda sygnał nie filtrowany (rys. 5.12) oraz sygnał oryginalny, to znaczy niezakłócony (rys. 5.13). Wystarczy, że w tym celu zaznaczysz odpowiednią opcję (**reference** albo **noisy**) grupy **Show signal** w obszarze prawego marginesu wyświetlanego przez program ekranu.

Popatrzmy teraz wspólnie, jak to działa. Pierwsze wyniki filtracji po wykonaniu niewielu kroków nie są zbyt obiecujące (rys. 5.14 i 5.15), jednak wytrwałe uczenie sieci (poprzez wielokrotne klikanie przycisku **Next**) prowadzi do tego, że w końcu dobrze wytrenowana sieć uczy się filtrować sygnał w sposób niemal idealny (rys. 5.16).

Proces uczenia może przebiegać automatycznie lub krok po kroku. Oznacza to, że na życzenie program pokaże Ci wszystkie swoje tajemnice albo może automatycznie pokonywać wiele dziesiątków etapów uczenia, podczas których sieć doskonali swoje działanie, a Ty spokojnie obejrzysz tylko efekt końcowy. O trybie procesu uczenia Ty decydujesz, w zależności od tego, którą z opcji grupy **Symulation mode** wybierzesz (**Auto** albo **Manual**). Na początku, w trakcie pierwszych eksperymentów, warto prześledzić krok po kroku, jak postępuje naprzód proces uczenia sieci, dlatego należy wybrać w tym celu tryb **Manual**. W kolejnych eksperymentach można zmodyfikować liczbę kroków (korzystając z opcji **Menu–>Configuration** na górnej krawędzi okna naszego programu) i spróbować dłuższego uczenia – na przykład 50 lub 100 kroków stosując tryb **Auto**. Wyniki są bardzo pouczające, warto więc się trochę potrudzić!

Jak widać z przedstawionych przykładów – sieć naprawdę uczy się i doskonali swoje działanie w taki sposób, że po pewnym (dość krótkim!) czasie stosunkowo skutecznie usuwa przypadkowe zakłócenia pojawiające się w oryginalnym sygnale. Skuteczność filtru, jaki powstaje w następstwie zastosowania procesu uczenia sieci, można ocenić, nakładając na przebieg sygnału przefiltrowanego – obraz sygnału przed filtracją (rys. 5.17).

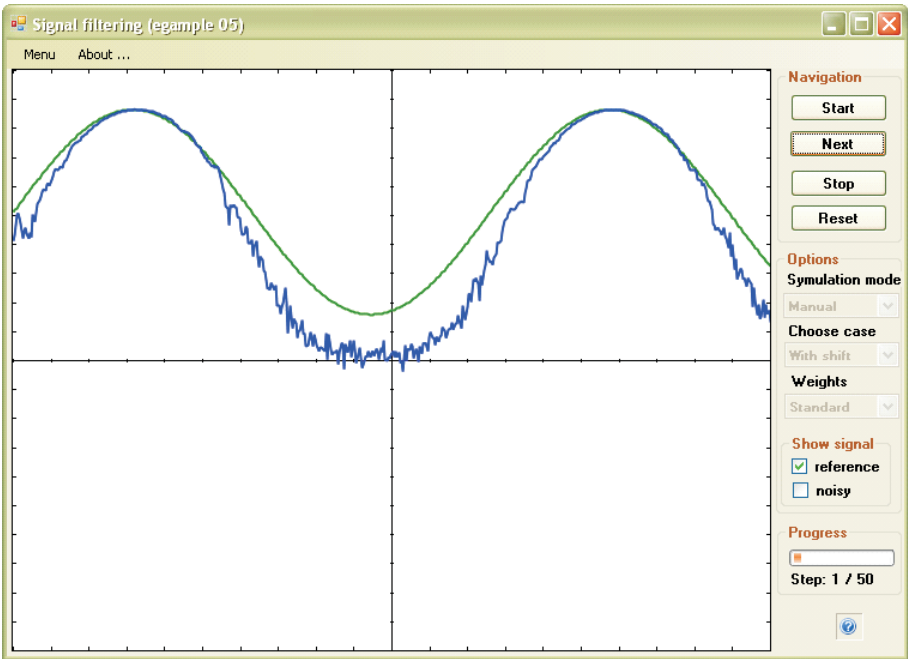



Rys. 5.17. Ocena skuteczności filtracji sygnału po dwudziestu krokach uczenia

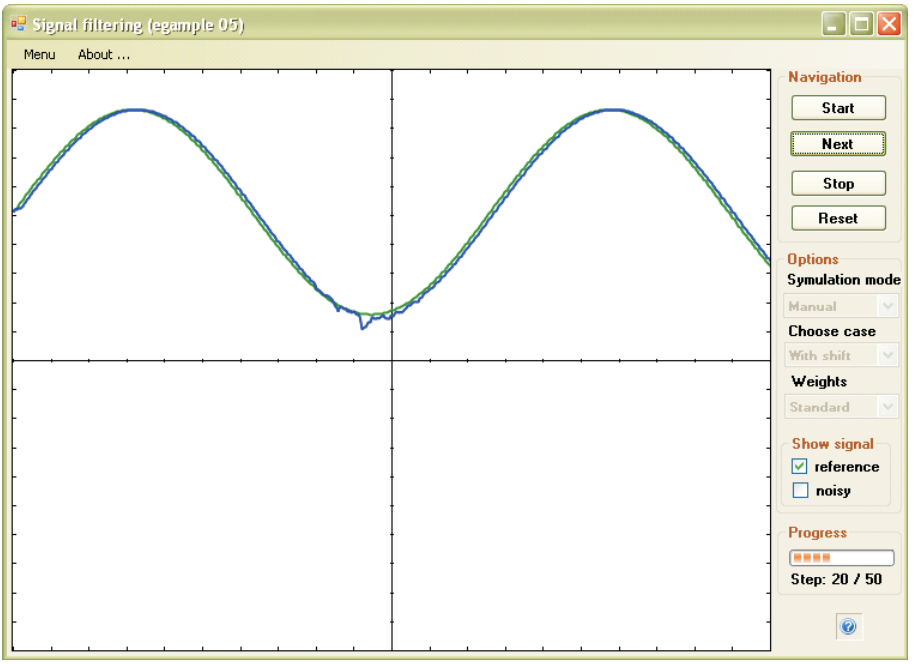
Jak się przekonasz eksperymentując z programem – sieć naprawdę potrafi nauczyć się filtracji sygnału i po pewnym czasie wykonuje to zadanie zupełnie dobrze. Łatwo też zauważysz, że najtrudniej jest nauczyć sieć odtwarzania sygnału w tych miejscach, w których wartości wzorcowego sygnału są małe (w szczególności wynoszą zero), dlatego w programie **Example 05** przewidziane są dwie wersje uczenia sieci – najpierw oryginalnym sygnałem sinusoidy, a potem sygnałem sinusoidy tak przesuniętej, by przetwarzane przez sieć wartości były wyłącznie dodatnie. Wybór wariantu wykonujesz za pomocą okienka „wybierz przypadek" (**Choose case** po prawej stronie). Na początku wyniki w tym drugim przypadku (określonym w okienku **Choose case** jako „z przesunięciem" – **With shift**) wydają się gorsze (patrz rys. 5.18), jednak wytrwałe uczenie sieci daje w tym przypadku znacznie lepsze wyniki niż wcześniej omówione (rys. 5.19).

Obserwacja zachowania sieci w obydwu rozważanych wariantach pomoże Ci w przyszłości wykrywać i analizować przyczyny ewentualnych niepowodzeń przy stosowaniu sieci do bardziej złożonych zadań.

Mam nadzieję, że przebieg procesu uczenia sieci filtrującej sygnały podobał Ci się i pomógł Ci zrozumieć, co i jak sieć neuronowa robi w trakcie doskonalenia swojego działania. Jednak przy całej atrakcyjności zadań, jakie




Rys. 5.18. Wynik filtracji sygnału przesuniętego po jednym kroku uczenia




Rys. 5.19. Wynik filtracji sygnału przesuniętego po dwudziestu krokach uczenia

w tym rozdziale wspólnie rozwiązaliśmy, muszę Ci coś zdradzić: sieci liniowe to dopiero przedszkole sieci neuronowych. Taka sobie rozgrzewka. Sieci te mogą bowiem być tylko jednowarstwowe (jeśli chcesz się dowiedzieć dlaczego – zajrzyj na koniec rozdziału 3 mojej książki pt. ***Sieci neuronowe***, tego nie da się wyjaśnić bez matematyki, a obiecywałem jej tu nie stosować), tymczasem kora mózgowa jest WIELOWARSTWOWA... Prawdziwa przygoda zacznie się więc dopiero wtedy, gdy zbudujesz i uruchomisz swoją pierwszą wielowarstwową sieć zbudowaną z nieliniowych neuronów. Jak się domyślasz, nastąpi to już niedługo – w kolejnym rozdziale...

## 5.8. Pytania i zadania do samodzielnego rozwiązania

1. Dlaczego przy uczeniu sieci bardziej skomplikowanych zadań stosujemy zbiór uczący zapisany na dysku, a nie uczymy sieci podając odpowiednie dane do programu za pomocą klawiatury albo myszki?
2. Jaka jest przyczyna tego, że wykres na rysunku 5.4 (i wiele innych wykresów zmian błędu, jakie zaobserwujesz w trakcie procesu uczenia) oprócz feralnej tendencji opadającej, oznaczającej postęp uczenia i systematyczne

malenie błędu, ma „wyskoki" do góry? Czy neuron od czasu do czasu nagle głupieje?

3. Spróbuj ocenić na podstawie doświadczeń opisanych w tym rozdziale, w jakim stopniu na końcowy wynik w postaci określonego zachowania się sieci neuronowej ma wpływ proces uczenia, a w jakim stopniu „wrodzone właściwości" wynikające z przypadkowej inicjalizacji parametrów występujących w sieci. Czy możesz z tego wyciągnąć praktyczne wnioski dotyczące wpływu Twojej edukacji na Twoją karierę życiową?

4. Spróbuj ustalić doświadczalnie (uruchamiając wiele razy program **Example 03**), jaka jest najkorzystniejsza wartość współczynnika uczenia (*Teaching ratio*) w zadaniu rozwiązywanym przez opisaną tutaj sieć. Jak sądzisz, czy przy innym zadaniu, opisanym całkiem innym zbiorem uczącym, optymalna wartość współczynnika uczenia będzie taka sama – czy może inna?

5. Zastanów się, kiedy można stosować większe wartości współczynnika uczenia (*Teaching ratio*): czy przy zadaniach łatwych do rozwiązania przez sieć neuronową, czy przy zadaniach trudnych i skomplikowanych?

6. Spróbuj wymyślić i zastosować inny zbiór uczący współpracujący z programem **Example 04,** zapisując nową zawartość w pliku **Default teaching set 04.txt** za pomocą Edytora **Notatnik** systemu Windows albo (lepiej) przy użyciu specjalizowanego narzędzia, jakim jest **Visual Studio.** Postaraj się doprowadzić do tego, żeby program **Example 04** mógł stanowić dla Ciebie uniwersalne narzędzie, pozwalające na rozwiązywanie różnych zadań, zależnie od swobodnie dobieranego przez Ciebie zbioru uczącego.

7. Przygotuj sobie kilka plików z różnymi zbiorami uczącymi i spróbuj porównać, jak dobrze uczy się sieć różnych zadań. Ustal, jak stopień trudności zadania (mierzony stopniem podobieństwa zbiorów danych opisujących te obiekty – zwłaszcza nietypowe – które mają być przez sieć **rozróżniane)** wpływa na czas uczenia i na poziom błędów, popełnianych przez sieć po zakończonym procesie uczenia. Postaraj się doprowadzić do sytuacji, w której zadanie stanie się tak trudne, że sieć nie zdoła się go nauczyć niezależnie od tego, jak długo będzie trwał proces uczenia.

8. Przebadaj, jak przebiega proces uczenia programu **Example 04** w zależności od początkowych wartości wag, współczynnika uczenia i różnych modyfikacji ciągu uczącego.

9. Sieć neuronowa ucząca się adaptacyjnej filtracji sygnału ma do dyspozycji sygnał zniekształcony szumem oraz wzorzec sygnału nie zakłóconego. Dlaczego przy uczeniu sieci nie korzysta się z wzorca samego sygnału zakłócającego (szumu)?

10. Na rysunkach 5.18 i 5.19 widać, że filtracja dokonywana przez program **Example 05** lepiej się udaje dla górnej części wykresu niż dla dolnej. Dlaczego?

**11. Zadanie dla zaawansowanych:** Program **Example 03** wykorzystywał uczenie sieci neuronowej do stworzenia **klasyfikatora** (sieci, która po podaniu określonego zestawu sygnałów wejściowych produkuje na wyjściach sygnał, który można interpretować jako **akceptację** obiektu opisanego danymi albo brak takiej akceptacji). Zbadaj, jak się zachowa ten program, gdy będzie zmuszony do wyuczenia się trudniejszej sztuki: obliczania na podstawie wartości danych wejściowych określonej **wartości** wyjściowej. Taka sieć będzie mogła pracować jako **model** jakiegoś prostego zjawiska fizycznego albo ekonomicznego. Oto przykładowy zestaw danych w formacie **Default teaching set.txt**, dla którego taki model możesz spróbować zbudować (przyjęto tylko dwa wejścia dla sieci, żeby nie trzeba było używać zbyt wielu danych uczących):

*2, 1*
*Observation 1*
*3, 4*
*–0.1*
*Observation 2*
*1, –2*
*0.7*
*Observation 3*
*4, 2*
*–0.2*
*Observation 4*
*0, –1*
*0.3*
*Observation 5*
*4, –5*
*1.9*
*Observation 6*
*–3, –3*
*0.6*
*Observation 7*
*–2, –4*
*1*
*Observation 8*
*3, –2*
*0.9*
*Observation 9*
*–1, –1*
*0.2*

Sprawdź, czy program prawidłowo podaje rozwiązania dla innych (nie używanych podczas uczenia) zestawów danych, wiedząc, że dane uczące były brane z problemu opisanego równaniem $y = 0.1\,X_1 - 0.3\,X_2$.

**12. Zadanie dla zaawansowanych:** Program **Example 05** jest programem ilustrującym sposób działania adaptacyjnego filtru opartego na zasadzie uczącej się sieci neuronowej, a nie programem roboczym, przeznaczonym do zastosowań praktycznych. Jednak taka sama sieć może służyć do filtracji innych sygnałów – na przykład zapisu **EKG.** Takiego sygnału w postaci cyfrowej pewnie nie masz do dyspozycji, ale spróbuj użyć odpowiednio zmodyfikowanej sieci do innych sygnałów – na przykład przefiltruj próbkę dźwięku w postaci pliku **WAV** albo **MP3**, a jeśli lubisz szczególnie trudne zadania, to pomyśl, jak podobną zasadę adaptacyjnej sieci filtrującej wykorzystać do przetwarzania obrazu.

# 6. Sieci nieliniowe

## 6.1. Po co komu nieliniowości w sieci?

Systemy liniowe (nie tylko sieci neuronowe – ogólnie wszelkie systemy tego typu) mają wiele sympatycznych właściwości. Ich zachowanie jest dobrze i łatwo przewidywalne, a matematyczny opis jest prosty i zawsze możliwy do rozwiązania. Czy jest więc sens porzucać ten tak wygodny i sympatyczny grunt i szukać szczęścia w trudnej i skomplikowanej dziedzinie sieci nieliniowych?

Otóż **j e s t** taka konieczność.

– Dlaczego?

Powodów jest kilka (i poznasz je wszystkie w dalszych częściach tego rozdziału), ale w tej chwili wymienię powód podstawowy: Po prostu **klasa zadań, jakie może rozwiązywać sieć zbudowana z nieliniowych neuronów, jest istotnie szersza niż klasa zadań, jakie rozwiązuje sieć liniowa.** Pierwsza sieć może realizować tylko takie zadania, które polegają na znalezieniu liniowego odwzorowania zbioru sygnałów wejściowych na zbiór sygnałów wyjściowych. Druga (to znaczy nieliniowa) sieć nie wykazuje podobnych ograniczeń.

Jeśli jesteś biegły w matematyce, to powyższe dwa zdania w zasadzie wszystko Ci wyjaśniły. Zakładam jednak, że matematyka niekoniecznie jest Twoją najmocniejszą stroną, dlatego zresztą całą tą książkę napisałem w taki sposób, żeby nie użyć w niej **ani jednej** formuły matematycznej. Skoro tak, to nie mogę Cię pozostawić w tej ważnej kwestii wyłącznie z tymi stwierdzeniami, które podałem wyżej, bo są one zapewne dla Ciebie pozbawione głębszej treści. No bo niby cóż takiego nadzwyczajnego ma wynikać z faktu, że oto sieć zbudowana z **liniowych** neuronów realizuje wyłącznie **liniowe** odwzorowania? Na pozór jest to trochę „masło maślane".

Gdybyś był biegły w matematyce, to mógłbym Ci naświetlić szerzej to zagadnienie stwierdzając, że odwzorowanie liniowe jest możliwe do przedstawienia w takiej formie, że rozwiązania dostarczane przez sieć neuronową (traktowane jako wektory) mogą być uzyskane z danych, podawanych do wejściowych neuronów sieci, za pomocą pewnej ustalonej **macierzy transformacji.** No tak, ale Tobie **macierz** kojarzy się zapewne z filmem ***Matrix***, więc to także zbyt wiele nie wyjaśnia.

Spróbuję więc scharakteryzować Ci przeciwstawienie **liniowych** oraz **nieliniowych** odwzorowań, odwołując się do ich fundamentalnych właściwości, nazywanych jednorodnością i addytywnością, chociaż nie musisz wcale starać się zapamiętać tych skomplikowanych nazw, tylko spróbuj sobie dobrze wyobrazić sens, jaki one niosą.

Jednorodność odwzorowania liniowego oznacza, że w tym odwzorowaniu skutek jest zawsze proporcjonalny do przyczyny. Niezależnie od tego, jakie pojęcia związesz z **przyczyną** i ze **skutkiem** – odwzorowanie, które pozwala Ci przewidzieć skutek, kiedy znasz przyczynę (albo odgadnąć przyczynę, kiedy zaobserwowałeś skutek), będzie **jednorodne**, jeśli na przykład dwukrotne zwiększenie przyczyny przyniesie dokładnie dwa razy większy skutek. **Proporcjonalne** zwiększanie skutku przy odpowiednim zwiększaniu przyczyny nie jest jednak cechą wszystkich możliwych zależności. Przeciwnie, obserwacja świata dostarcza Ci mnóstwa przykładów zależności, w których taka proporcjonalność nie zachodzi. Na przykład, wiesz dobrze ze swojej szkolnej praktyki, że jeśli więcej się uczysz, to otrzymujesz lepsze oceny. Zatem między czasem poświęconym na naukę a wartością oceny, jaką Ci wystawi nauczyciel, jest jakaś zależność, którą można przedstawić jako pewne odwzorowanie. Jednak z pewnością **nie jest prawdą,** że gdy dwa razy dłużej będziesz się uczył, to otrzymasz dokładnie dwa razy lepszą ocenę! Wobec tego zależność między wysiłkiem, jaki wkładasz w naukę, a ocenami, jakie dostajesz od nauczycieli, jest **nieliniowa** bo odpowiednie odwzorowanie nie zachowuje własności jednorodności, a jednorodność jest warunkiem koniecznym liniowości.

Drugim warunkiem liniowości jest addytywność. Ten trudno brzmiący termin też ma proste wytłumaczenie. Otóż jeśli badasz rozważaną zależność, którą ma opisywać interesujące Cię odwzorowanie, to możesz sprawdzić, jaki skutek otrzymasz po zastosowaniu jakiejś jednej przyczyny, a potem możesz tak samo zbadać, jaki skutek uzyskasz po zastosowaniu jakiejś innej przyczyny. Znając skutek uzyskany oddzielnie dla pierwszej przyczyny i oddzielnie dla drugiej przyczyny – usiłujemy przewidzieć, jaki będzie skutek, gdy obie przyczyny zadziałają razem. Jeśli skutkiem sumarycznego działania obu przyczyn będzie efekt, będący dokładnie **sumą** skutków działania obydwu

tych przyczyn oddzielnie – to mówimy, że badane zjawisko (i opisujące je odwzorowanie) jest **addytywne.** Niestety, znowu można pokazać bardzo dużo takich zjawisk i procesów, w których takie proste sumowanie skutków w przypadku sumowania przyczyn – po prostu nie zachodzi. Jeśli napełnisz dzban wodą – to będziesz mógł ułożyć w nim kwiaty. Jeśli zrzucisz go ze stołu – to będzie leżał na podłodze. Ale jeśli napełnisz dzban wodą i zrzucisz go ze stołu, to nie uzyskasz możliwości ułożenia w nim bukietu, który będzie stał na podłodze – bo gdy dzban pełen wody spadnie ze stołu, to się na pewno potłucze. Zjawisko stłuczenia dzbana jest więc zjawiskiem nieliniowym i jako skutek nie da się przewidzieć jako prosta suma skutków uprzednich działań rozpatrywanych oddzielnie.

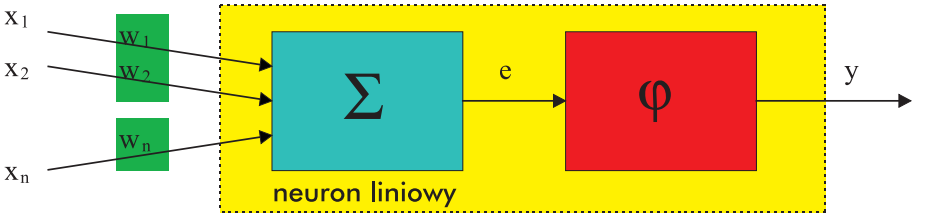
Gdy budujemy matematyczne opisy jakichś systemów, procesów albo zjawisk – to chętnie używamy opisów liniowych, bo są łatwiejsze do zastosowania. Jeśli te matematyczne opisy chcemy odwzorować w sieci neuronowej – to jest nam miło, gdy może to być prosta i sympatyczna sieć liniowa.

Niestety, bardzo dużo systemów, procesów albo zjawisk nie da się „wepchnąć" w ramy odwzorowań liniowych. Dla nich musimy stosować skomplikowane nieliniowe modele matematyczne albo – łatwiej i wygodniej – możemy zbudować nieliniowe sieci neuronowe, które takie systemy, procesy albo zjawiska będą wiernie modelować. Sieć neuronowa jest w takich okolicznościach niezwykle potężnym i niezwykle skutecznym narzędziem. Charakter zależności wiążącej sygnały wyjściowe (odpowiedzi sieci) z sygnałami wejściowymi (to znaczy danymi będącymi punktem wyjścia do obliczeń) przy odpowiednio dużej, wielowarstwowej sieci nieliniowej może być – praktycznie dowolny. Wynika to między innymi z jednego z fundamentalnych twierdzeń matematycznych na temat interpolacji i ekstrapolacji funkcji, związanego z nazwiskiem wielkiego rosyjskiego matematyka Kołmogorowa.

Nie jest to właściwe miejsce, by temat ten, mający wyraźnie teoretyczny i matematyczny charakter, jeszcze dokładniej przedstawiać i szczegółowo drążyć – proponuję więc chwilowo przyjąć to stwierdzenie „na wiarę". Pragnę Cię tylko zapewnić, że zagadnienie wyższości sieci nieliniowych nad liniowymi nie ma wyłącznie charakteru akademickiego (typu pamiętnych „naukowych" dyskusji *o wyższości Świąt Wielkiej Nocy nad Świętami Bożego Narodzenia*), lecz wiąże się bezpośrednio z oceną praktycznej użyteczności rozważanych tu narzędzi. Naprawdę istnieje spora grupa na wskroś **praktycznych** zadań, które mogą być rozwiązane **tylko** za pomocą sieci nieliniowej – a za pomocą liniowej nie. Jeden z programów, które w tym rozdziale będziesz badał, zilustruje to zagadnienie nieco dokładniej, ale najpierw wypróbuj kilka innych programów, pokazujących, jak taka nieliniowa sieć działa i jakie są jej możliwości.

## 6.2. Jak działa nieliniowy neuron?

Zaczniemy od prostego programiku, pokazującego działanie **jednego** nieliniowego neuronu. Schemat takiego neuronu przypominam Ci na rysunku 6.1.

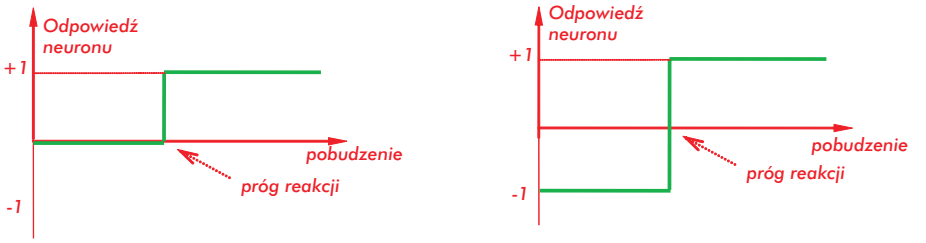



6.1. Struktura neuronu nieliniowego jako rozszerzenie neuronu liniowego o nieliniową funkcję przejścia

Program modelujący ten neuron (**Example 06a**) jest analogiczny (z pewnymi uproszczeniami) do programu **Example 01c**, za pomocą którego badałeś poprzednio działanie neuronu liniowego. Przykład ten pomoże Ci prześledzić, jak działa prosty neuron o nieliniowej (w tym przypadku – progowej) charakterystyce. Program przewiduje możliwość korzystania z tak zwanej charakterystyki *unipolarnej* albo *bipolarnej*. Poprosi Cię więc zaraz na początku o wybór jednej z nich, patrz rys. 6.3a – pole **Neuron type** (dodatkowo możesz ustalić liczbę wejść do neuronu, ale na początku radzę Ci zaakceptować podaną domyślnie liczę 4 i klikając **Next** przejść do kolejnego okna). Wbrew temu, co mógłbyś przypuszczać na podstawie skomplikowanego brzmienia odpowiednich nazw – sprawa jest w istocie bardzo prosta: przy charakterystyce unipolarnej sygnał wyjściowy neuronu jest zawsze nieujemny (najczęściej przyjmuje wartości **1** oraz **0,** ale niektóre typy nieliniowych neuronów mogą też produkować sygnały o dowolnych wartościach pośrednich pomiędzy tymi dwiema skrajnościami), natomiast przy charakterystyce bipolarnej możliwe są zarówno dodatnie, jak i ujemne wartości sygnałów (najczęściej są to wartości **+1** oraz **–1,** ale tu także dopuszcza się wartości pośrednie). Porównanie charakterystyki bipolarnej i unipolarnej znajdziesz także na rysunku 6.2, możesz więc poglądowo zorientować się, na czym w istocie polega różnica. Dodam tylko jedno.

Prawdziwe, biologiczne neurony nie znają pojęcia „ujemny sygnał". Wielkości wszystkich sygnałów w Twoim mózgu są wyłącznie dodatnie (lub zerowe, gdy się lenisz!), zatem bliższa biologicznej rzeczywistości jest zdecydowanie charakterystyka unipolarna. Jednak w przypadku technicznych sieci neuronowych zwykle bardziej staramy się o uzyskanie wygodnego narzędzia obliczeniowego niż o zachowanie maksymalnej wierności dla biologicznego

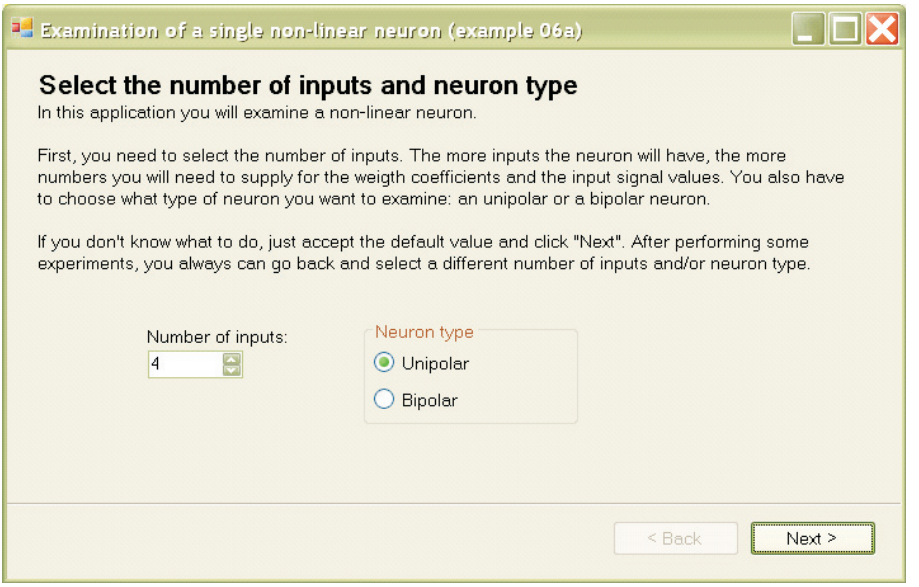
pierwowzoru. Dlatego również w odniesieniu do charakterystyk używanych w technice neuronów dokonujemy zwykle pewnego „nadużycia". Polega ono na tym, że staramy się uniknąć sytuacji, w której w sieci pojawiałyby się sygnały o wartości **0** – zwłaszcza jeśli odbiorcami sygnałów generowanych przez jeden neuron mają być inne neurony tej samej sieci. Jak być może pamiętasz – sieć kiepsko się uczy, jeśli ma do czynienia z sygnałami przyjmującymi wartość **0**. (Pamiętasz może zabawę w filtrację sygnałów za pomocą sieci, którą Ci zaproponowałem w poprzednim rozdziale? Tam było to widać jak na dłoni!). W związku z tym obok sieci złożonej z elementów pracujących z sygnałami **0** i **1** wprowadza się w technice sieci neuronowych struktury **bipolarne**. W takich neuronach też mamy dwa rodzaje sygnałów, ale oznaczone są one jako **–1** i **+1.** Proste, prawda? No to obejrzyj to jeszcze dokładnie na rysunku 6.2.



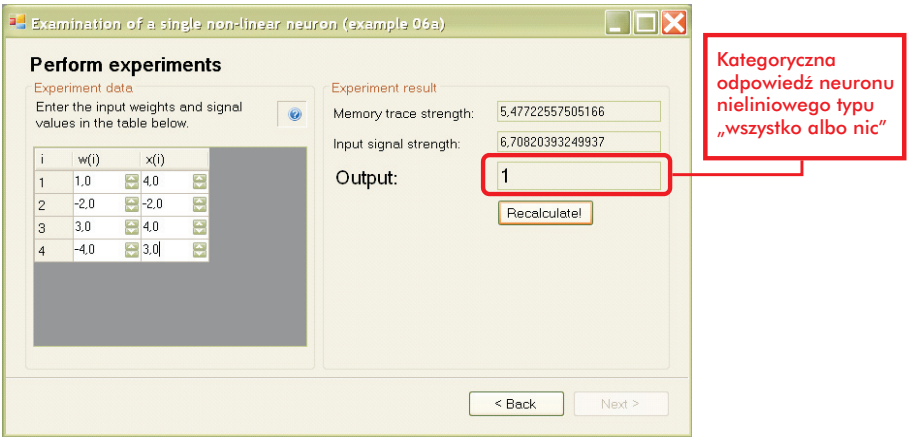

6.2. Nieliniowe charakterystyki neuronu: unipolarna (po lewej) i bipolarna (po prawej)

Używanie programu **Example 06a** jest proste i intuicyjnie oczywiste. Zauważysz zapewne, bawiąc się tym programem, że działa on bardziej kategorycznie od poznanego wcześniej neuronu liniowego (por. rys. 6.3b). Tamte wcześniej rozważane neurony (i zbudowane z nich sieci) reagują na podawane do nich sygnały w sposób subtelny i wyważony: jedne kombinacje sygnałów wejściowych powodują silne reakcje (duży sygnał wyjściowy), inne wywołują reakcje zdecydowanie słabsze, a jeszcze inne traktowane są w sposób prawie zupełnie obojętny (sygnał wyjściowy jest bliski zera). W odróżnieniu od tych subtelnych neuronów liniowych – neurony nieliniowe działają na zasadzie *„wszystko albo nic"*. Na pewną kombinację sygnałów neuron może reagować zdecydowanie negatywnie (sygnał wyjściowy wynosi –1), ale wystarczy czasem minimalna zmiana sygnałów wejściowych, by odpowiedź neuronu stała się zdecydowanie i jednoznacznie pozytywna (sygnał wyjściowy zmienia się na 0 lub +1) – rys. 6.4.

Opisany wyżej program **Example 06a** jest bardzo prymitywny i ubogi. A jednak jest to model – z dotychczas przez nas rozważanych – najbardziej podobny do tej techniki i tej zasady działania, jaką odnaleźć można w rzeczy-

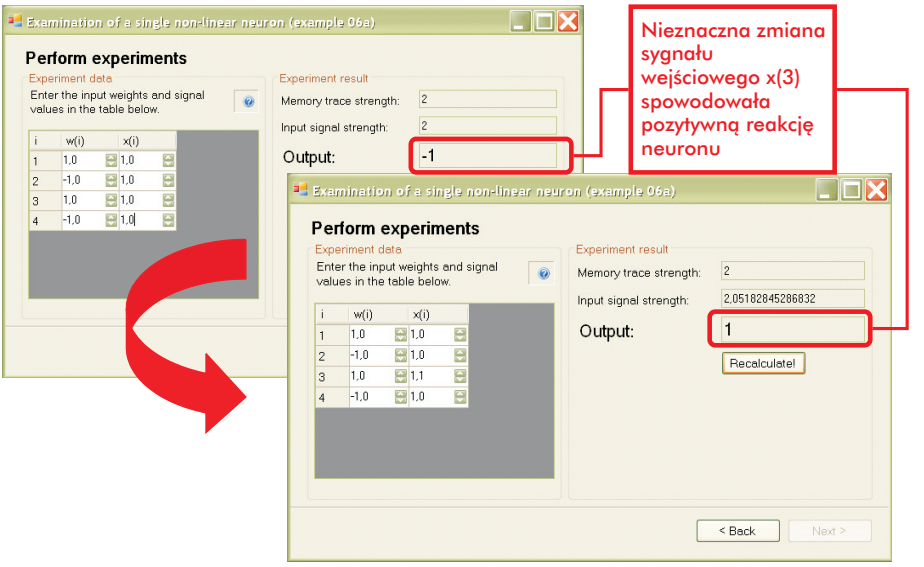



Rys. 6.3a. Początek konwersacji z programem **Example 06a**




Rys. 6.3b. Dalszy etap konwersacji z programem **Example 06a**

wistym ludzkim mózgu. Kilka prostych eksperymentów, podobnych do tych, jakie wcześniej wykonywałeś z neuronami liniowymi, pozwoli Ci zorientować się, jak ten program działa. Niby nic wielkiego ani szczególnie ciekawego. Ot, taka sobie prosta funkcja. A jednak przy odrobinie fantazji możesz sobie wyobrazić, że oto eksperymentujesz na rzeczywistej komórce nerwowej...




Rys. 6.4. Wrażliwość nieliniowego neuronu na zmianę sygnałów wejściowych

*...W niklowanym uchwycie aparatu widnieje otwarta czaszka, a w jej głębi tętni w rytm uderzeń serca szara, pofałdowana powierzchnia mózgu. Mikroskopijna elektroda zanurza się w tkance i na monitorach zaczynają się pojawiać charakterystyczne kształty sygnałów nerwowych – zawsze takich samych stromych jak iglice impulsów, identycznych we wszystkich obszarach mózgu i we wszystkich sytuacjach – a jednak znaczących wciąż coś innego.*

*Drobny ruch gałkami stymulatorów wysyłających w głąb tkanki elektryczne sygnały – i impulsy znikają. Trochę inna kombinacja sygnałów – i impuls znowu jest. Trudne do uwierzenia, ale prawdziwe – tak właśnie działa rzeczywisty, ludzki mózg.* ***Twój*** *mózg. I to właśnie w tej chwili. W trylionach komórek pojawiają się i nikną elektryczne impulsy.* ***Tylko*** *tyle i* ***aż*** *tyle.*

*W tym mieści się wszystko: Rozwiązanie równania i wielka miłość. Zrozumienie istoty bytu i chęć popełnienia najpodlejszej zbrodni. Zapach łąki na wiosnę i widok gwiaździstego nieba. Wszystkie nie wypowiedziane słowa i ukryte pragnienia. Pomysł, który przyszedł Ci do głowy przed chwilą i wspomnienie z dzieciństwa, które wraca w snach... Impulsy, tylko impulsy – albo ich brak. Wszystko – albo nic.* ***1*** *lub* ***0****. Jak w komputerze...*

Fantazja to rzecz piękna i inspirująca, nie rozwija jednak wiedzy, wróćmy zatem do rzeczywistości i do dalszych eksperymentów.

## 6.3. Jak działa sieć złożona z nieliniowych neuronów?

W przytoczonym wyżej programie używałeś neuronu, którego wyjście przyjmowało dwie wartości, które mogły być kojarzone z akceptacją (rozpoznaniem) pewnego zestawu sygnałów, lub z jego całkowitym odrzuceniem. Taka akceptacja lub odrzucenie wiąże się często z **rozpoznaniem** jakiegoś obiektu lub jakiejś sytuacji, dlatego sieci złożone z takich neuronów, jaki badałeś w programie **Example 06a**, nazywa się często **perceptronami**.

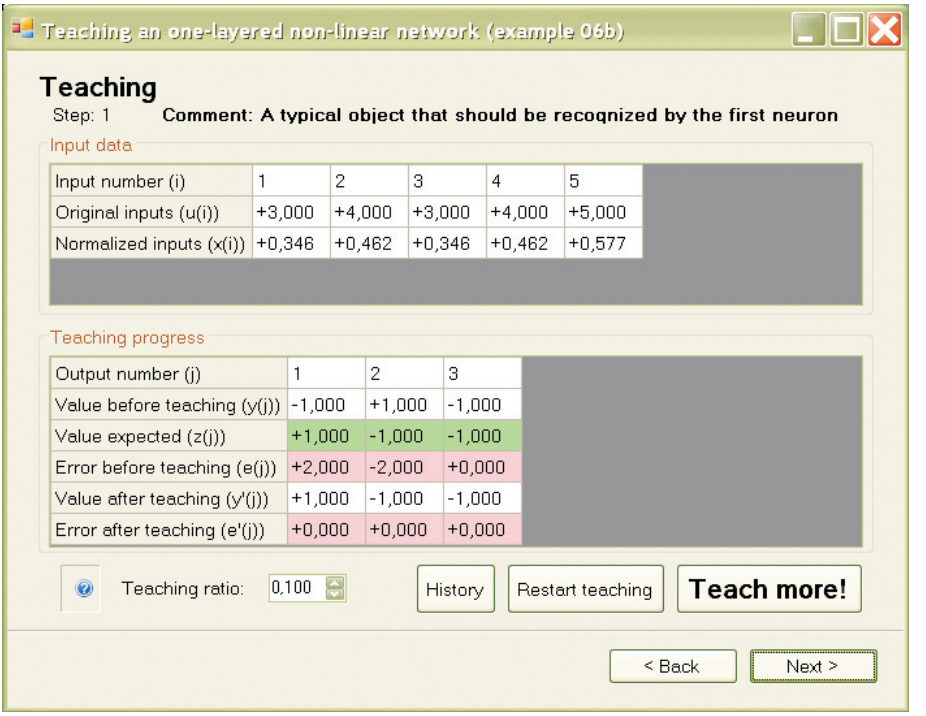
Neurony typu perceptronowego podają na swoich wyjściach jedynie dwa rodzaje sygnałów: **1** albo **0**. Odpowiada to – w pewnym stopniu oczywiście – znanej z neurofizjologii zasadzie *„wszystko albo nic”*, odnoszącej się do typowego działania rzeczywistego biologicznego neuronu. Tak działa opisany wyżej program **Example 06a** i z takich neuronów będziesz budował wszystkie dalsze, opisywane w tej książce sieci. Teraz prześledzisz działanie i uczenie się zespołu neuronów bipolarnych na przykładzie prostej jednowarstwowej sieci nieliniowej złożonej z takich właśnie neuronów. Sieć taką opisuje kolejny program **Example 06b**. Dla uczenia sieci z tego programu możesz wykorzystać plik o nazwie **Default teaching set 06b.txt,** proponowany domyślnie przez **Example 06b** i zawierający przykłady, na których będziesz uczył swoją sieć, co ma robić. Plik ten może mieć na przykład taką zawartość:

```
5, 3
A typical object that should be recognized by the first neuron
3, 4, 3, 4, 5
1, –1, –1
A typical object that should be recognized by the second neuron
1, –2, 1, –2, –4
–1, 1, –1
A typical object that should be recognized by the third neuron
–3, 2, –5, 3, 1
–1, –1, 1
```

Zawartość podobnego pliku była już wcześniej omawiana, więc wydaje się, że nie ma już potrzeby szczegółowego powtórnego omawiania jego zawartości – jeśli jednak masz tu jakieś wątpliwości, to zajrzyj do poprzedniego rozdziału.

Porównując podany wyżej plik z tymi, których używałeś wcześniej, zauważ, że wystarczy tak niewielka liczba przykładów, gdyż – o czym się przekonasz, gdy tylko uruchomisz program – sieć nieliniowa potrafi się bardzo

szybko uczyć. Zazwyczaj[1] wystarczy już jeden krok uczenia, by zlikwidować pojawiające się błędy (rys. 6.5).

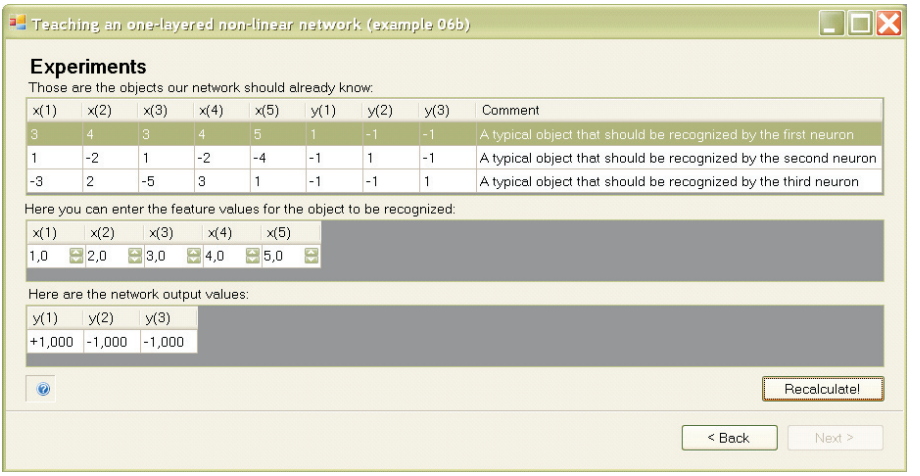



Rys. 6.5. Już w pierwszym kroku uczenia nieliniowej sieci udaje się usunąć większość błędów

Przeciętnie po 6–7 krokach sieć jest całkowicie nauczona i można przystępować do egzaminu. Egzamin ten dowodzi, że sieć nie tylko szybko się uczy, ale i sympatycznie uogólnia zdobytą wiedzę (rys. 6.6).

Zjawisko uogólniania wiedzy jest w sieciach neuronowych zawsze bardzo potrzebne i bardzo ważne, bo przecież mały byłby pożytek z sieci neuronowej, którą byś najpierw nauczył, pokazując jej szereg zadań wraz z ich poprawnymi rozwiązaniami, a potem byś stwierdził, że sieć umie już rozwiązywać zadania – ale tylko te, które jej wcześniej pokazałeś. Przecież tych zadań, które są podawane w zbiorze uczącym, rozwiązywać nie trzeba, bo dla nich rozwiązania są już znane. Tym, czego naprawdę potrzebujesz, jest narzę-

[1] Warunkowa forma tego stwierdzenia wynika z faktu, że w sieciach neuronowych nigdy niczego nie można być tak do końca pewnym – każda próba, każdy krok procesu uczenia może być inny, bo na ostateczny wynik mają silny wpływ czynniki losowe, które nigdy nie są dwa razy dokładnie takie same. Zupełnie jak w życiu, jak w **Twoim** życiu!




Rys. 6.6. Sieć nieliniowa dobrze uogólnia zdobytą w trakcie uczenia wiedzę

dzie, które najpierw nauczy się rozwiązywać zadania ze zbioru uczącego, ale potem będzie także rozwiązywać **inne** zadania. Na szczęście sieć neuronowa potrafi **uogólniać** wiedzę, rozwiązując sprawnie także takie zadania, które nie występowały w zbiorze uczącym, ale które są do tych ***uczących*** w jakimś sensie **podobne** (opierają się na podobnej logice zależności między wejściem i wyjściem).

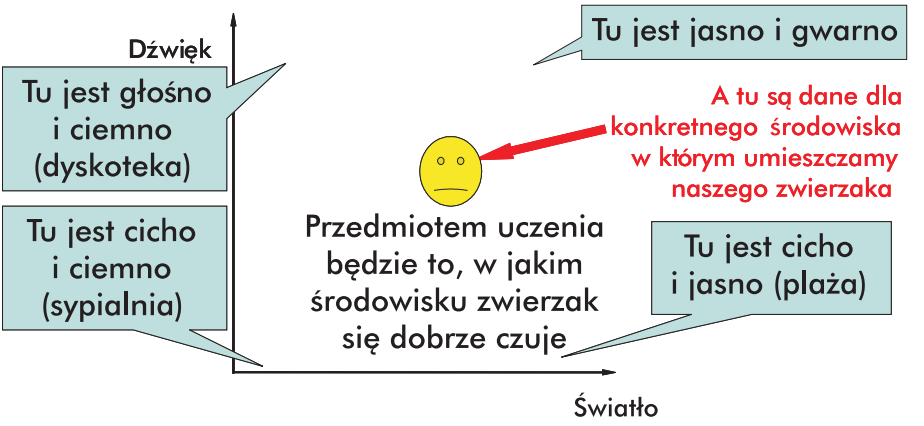
Właściwość uogólniania wiedzy jest jedną z najważniejszych cech sieci neuronowej, dlatego namawiam Cię, żebyś sprawdził, jak dalece można zniekształcić dane wejściowe w stosunku do tych danych, na bazie których sieć była nauczana, żeby wyniki podawane przez sieć były nadal sensowne i mogły być uznane za efekt uogólniania wiedzy, a nie całkowicie swobodnego fantazjowania – do którego sieć jest także zdolna! Gorąco polecam Ci, żebyś sam spróbował jeszcze na wiele innych sposobów „poznęcać się" nad siecią – a z pewnością uznasz jej wyższość nad sieciami liniowymi!

## 6.4. Jak przedstawić działanie nieliniowych neuronów?

Wyniki działania nieliniowych neuronów i budowanych z nich sieci wygodnie jest interpretować w taki sposób, że rysuje się tak zwaną ***przestrzeń sygnałów wejściowych*** i pokazuje się, dla których wartości tych sygnałów neuron reaguje pozytywnie (to znaczy wysyła sygnał +1), a dla których negatywnie (to znaczy wysyła sygnał –1).

Zastanówmy się wspólnie, jak interpretować dane pokazywane na rysunku przestrzeni sygnałów wejściowych?

Otóż **każdy punkt wewnątrz dużego kwadratu,** który za chwilę narysuje Ci program **Example 06c,** symbolizuje zespół dwóch danych (odpowiadających poziomej i pionowej współrzędnej położenia punktu), stanowiących aktualne sygnały wejściowe dla sieci neuronowej. Możesz to sobie wyobrazić na przykład w ten sposób, że badana sieć jest mózgiem hipotetycznego zwierzęcia wyposażonego w dwa receptory – na przykład prymitywny wzrok i słuch. Im silniejszy jest sygnał odbierany przez wzrok – tym bardziej na prawo znajduje się punkt. Im silniejszy jest dźwięk – tym wyżej będzie się znajdował punkt na obrazku (rys. 6.7).

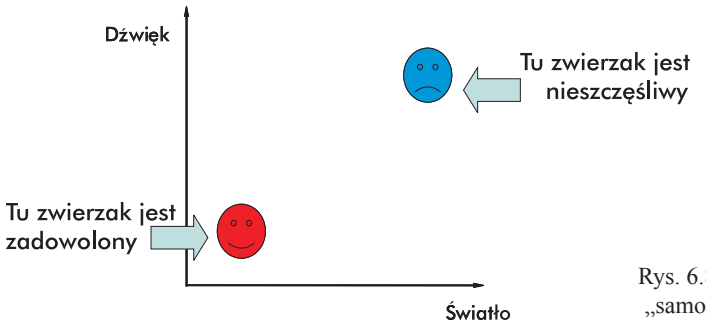



Rys. 6.7. Eksperymentalny świat zwierzaka, którego mózgiem jest badany neuron

Jak z tego wynika, lewy dolny róg kwadratu odpowiada sytuacji ciszy i ciemności (*sypialnia*), a prawy górny – otoczeniu maksymalnie głośnemu i maksymalnie jasno oświetlonemu (*Rynek Główny* w południe). Lewy górny róg to może być *dyskoteka* (ciemno i głośno), a prawy dolny to *plaża* (cisza i słońce). Analogie dla pozostałych punktów wewnątrz kwadratów spróbuj sam sobie wyobrazić.

Do każdej z omówionych sytuacji modelowane „zwierzę" może mieć nastawienie pozytywne (punkt wtedy będzie miał barwę intensywnie **czerwoną**) lub negatywne (punkt będzie miał kolor **błękitny** – patrz rys. 6.8).

Wyobraź sobie teraz, że „mózg" modelowanego przez Ciebie zwierzęcia zawiera tylko jeden, ale za to nieliniowy neuron. Musi to być oczywiście neuron o dwóch wejściach, ponieważ rozważasz sytuację, kiedy informacje wejściowe pochodzą z dwóch receptorów „wzroku" i „słuchu". Dla każdego z tych receptorów najpierw dowolnie określisz współczynniki wagowe – na przykład podając dodatni pierwszy współczynnik wagi wskazujesz, że two-



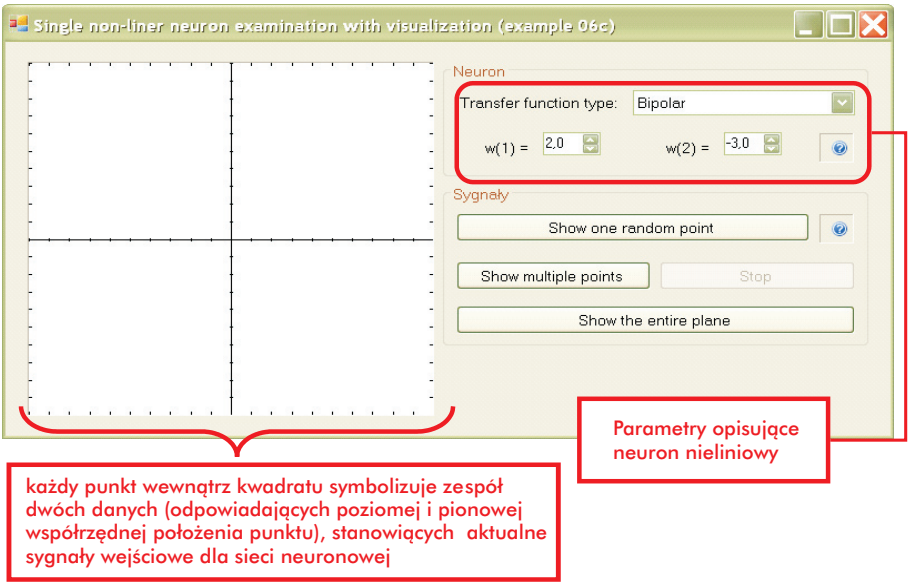

Rys. 6.8. Sposób oznaczania „samopoczucia" zwierzaka

je „zwierzę" ma lubić jasne oświetlenie, a podając w tym miejscu wartość ujemną skłaniasz je do tego, by chętnie kryło się w ciemnym kącie. Podobnie możesz postąpić z drugą współrzędną wektora wag, określając, czy Twoje zwierzę woli ciszę, czy jest zwolennikiem koncertów zespołu *Metalica.*

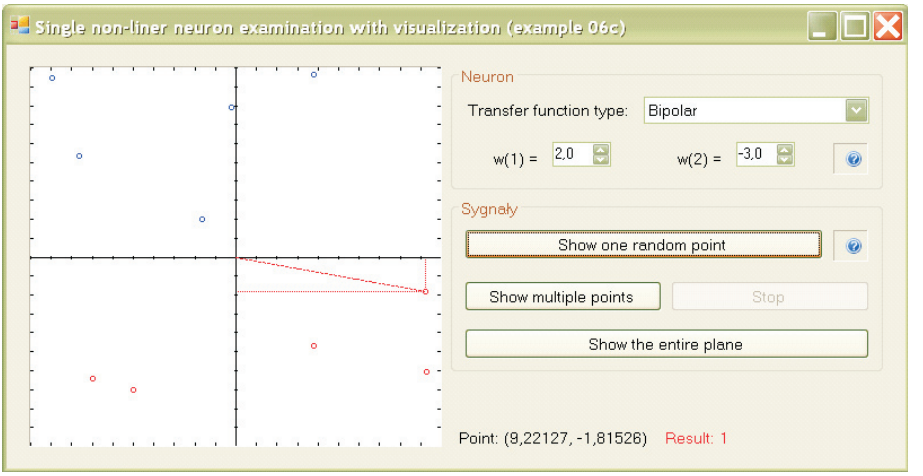
W ten sposób „zaprogramujesz" mózg swojego „zwierzęcia", który potem będziesz testował, podając na jego wejście różne kombinacje wejściowych sygnałów (oczywiście także dwóch). Każda podana na wejście kombinacja sygnałów może być traktowana jako określone „środowisko", w którym umieścisz na próbę swoje „zwierzę". Właściwości tego „środowiska" symbolizuje punkt na płaszczyźnie rysunku – wartość pierwszego sygnału (oświetlenie) wyznacza odciętą tego punktu, a wartość drugiego sygnału (dźwięk) – jego rzędną. Dla takiego punktu (to znaczy dla takiego zestawu sygnałów wejściowych) neuron ustala swoją decyzję – w jednym wypadku będzie to +1, a w drugim –1. Innymi słowy, w jednych warunkach „zwierzęciu" się podoba, a w innych nie. Zrób mapę obrazującą preferencje Twojego neuronu kolorując na **czerwono** te punkty, które neuron „zaakceptował" (wysłał na swoim wyjściu sygnał **+1**), a na **niebiesko** te punkty, które neuron „odrzucił" (sygnalizując **–1**)... A właściwie po co masz to robić sam – lepiej skorzystaj z mojego programu **Example 06c**, który zrobi to za Ciebie. Najpierw oczywiście musisz podać współczynniki wag, definiujące, co Twój neuron lubi (rys. 6.9).

Potem możesz już do woli sprawdzać, jaką wartość (+1 albo –1) wytwarza na swoim wyjściu neuron w określonych punktach swojej przestrzeni sygnałów wejściowych. Program jest dość usłużny – nie dość, że wykonuje skwapliwie wszystkie potrzebne obliczenia i rysuje opisaną wyżej mapę, to jeszcze grzecznie sam generuje potrzebne sygnały wejściowe dla neuronu i podaje ich współrzędne u dołu okienka (po prawej stronie – patrz rys. 6.10).

Dopóki masz ochotę obserwować położenie każdego kolejno prezentowanego punktu – możesz naciskać przycisk **Show one random point** i sprawdzać położenie nowego punktu na ekranie (aktualnie wygenerowany punkt jest zaznaczony poprzez rzutowanie jego współrzędnych na osie, więc można



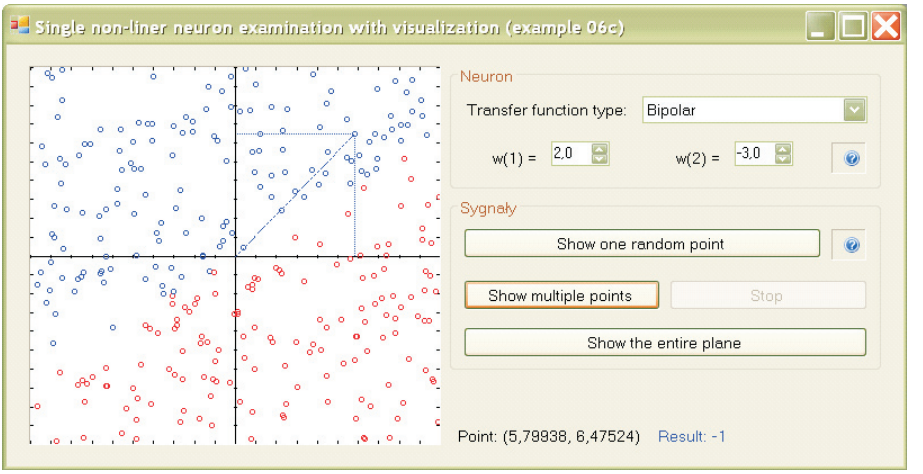

Rys. 6.9. Ustawianie parametrów neuronu w programie **Example 06c**




Rys. 6.10. Prezentacja (za pomocą koloru) odpowiedzi neuronu na podany sygnał wejściowy

go łatwo odróżnić od innych, które pojawiły się wcześniej). Czyniąc tak, możesz się zastanawiać, dlaczego neuron zaakceptował go lub odrzucił. Jest to łatwe, ponieważ dane na temat aktualnie rozważanego punktu uwidocznione są dodatkowo liczbowo w linii poniżej rysunku. Ponieważ jednak punktów będzie potrzeba dużo, żeby skutecznie zrobić mapę dokładnie pokazującą,

w których obszarach przestrzeni sygnałów wejściowych neuron odpowiada pozytywnie, a w których negatywnie – możesz w pewnym momencie nacisnąć przycisk **Show multiple points**. Spowoduje to przejście programu w tryb pracy ciągłej, w czasie której kolejne punkty są automatycznie generowane i wyświetlane na „mapie" (zawsze możesz powrócić do poprzedniego trybu przerywając proces przyciskiem **Stop**). Po pewnym czasie ukaże się zarys obszaru, w którym neuron odpowiada pozytywnie, oraz drugiego obszaru, w którym odpowiedź neuronu jest negatywna (rys. 6.11).




Rys. 6.11. Widok ekranu programu **Example 06c** po wielu pokazach

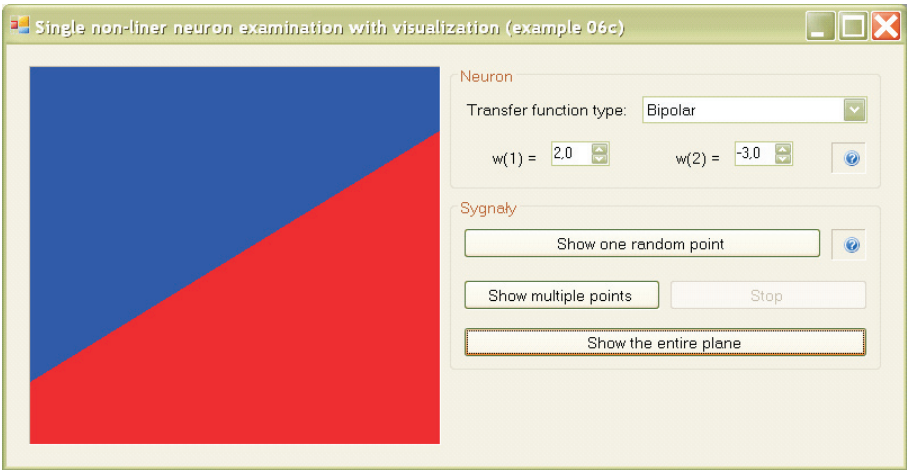
Granicą pomiędzy tymi obszarami jest – co łatwo uzasadnić – linia prosta, mówimy więc czasem, że pojedynczy neuron **dokonuje dyskryminacji liniowej przestrzeni wejść** – brzmi to mądrze i tajemniczo, a oznacza dokładnie to, co widziałeś na ekranie w czasie prowadzenia opisanych tu eksperymentów.

Jeśli chcesz zobaczyć, jak wyglądałaby mapa odpowiedzi neuronu bez zbytniego wysiłku włożonego w generowanie punktów, to program **Example 06c** udostępnia Ci również i taką możliwość. W tym celu wykorzystaj przycisk **Show the entire plane** (rys. 6.12). Ciekawym doświadczeniem może być również zmiana nieliniowej charakterystyki neuronu poprzez wybór jednej z dostępnych na liście **Transfer function type** (w grupie opcji związanej z neuronem), do czego gorąco Cię zachęcam.

## 6.5. Jakie możliwości ma wielowarstwowa sieć nieliniowych neuronów?

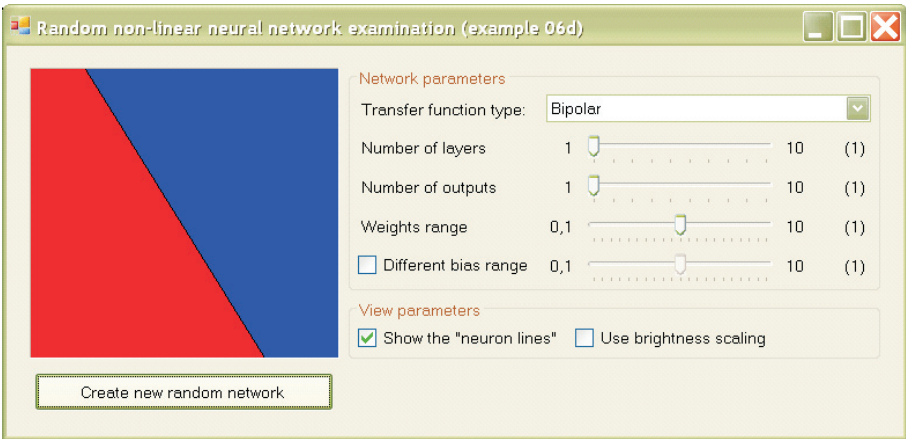
Jeden neuron (albo jedna warstwa neuronów) może tylko tak właśnie rozdzielać sygnały wejściowe – linią graniczną (w dwuwymiarowej przestrzeni, czyli przy dwóch sygnałach wejściowych wprowadzanych do sieci) jest zawsze linia prosta. Odpowiednio kombinując ze sobą takie liniowo rozgraniczone obszary i odpowiednio je ze sobą zestawiając – można uzyskać właściwie dowolne wartości zagregowanych sygnałów wyjściowych w dowolnych punktach przestrzeni sygnałów wejściowych. Stąd w literaturze dotyczącej sieci neuronowych często pojawiają się nieco mylące twierdzenia, że za pomocą neuronów można uzyskać dowolnie dokładną aproksymację **dowolnej** funkcji. W istocie nie da się tego zrobić na jednej warstwie neuronów, bo twierdzenia Cybenki, Hornika, Lapedesa i Farbera oraz wiele innych podobnych naukowych wywodów w istocie odwołują się do sieci o większej liczbie warstw. Użycie wielu warstw jest konieczne, gdyż elementy strukturalne tworzące wymaganą funkcję, zbudowane przez uczące się neurony, muszą być na końcu **zebrane** przez przynajmniej jeden neuron wynikowy (najczęściej o charakterystyce liniowej). Fakt ten powoduje, że sieci opisywane jako uniwersalne aproksymatory, mają w istocie więcej warstw – przynajmniej dwie, a często trzy. W dodatku takie zuchwałe twierdzenia mówiące, że sieci mogą aproksymować **dowolną** funkcję, drażnią matematyków, którzy dokładają wtedy starań, żeby wymyślić taką dziwną funkcję, aby jej żadna sieć neuronowa „nie ugryzła". Oczywiście, okazuje się, że matematycy mają rację – da się wymyślić tak paskudną funkcję (i to nie jedną!), że sieć neuronowa nie ma szans. Ale Przyroda jest łaskawsza od matematyków. Rzeczywiste procesy (fizyczne, chemiczne, biologiczne, techniczne, ekonomiczne, społeczne i inne pochodzące z realnego świata) dają się opisywać funkcjami o pewnym stopniu regularności – a takie sieci neuronowe są w stanie obsłużyć bez większego trudu, zwłaszcza jeśli zaangażuje się więcej warstw.

Jest bowiem rzeczą bezsporną, że sieć złożona z wielu warstw neuronów ma znacznie bogatsze możliwości niż omawiana wyżej sieć jednowarstwowa. Tu obszary, w których neurony będą pozytywnie reagowały na sygnały wejściowe, mogą mieć bardziej skomplikowaną formę. W przypadku sieci dwuwarstwowej będą to pewne wielokąty wypukłe (lub ich wielowymiarowe odpowiedniki, tak zwane *simpleksy*), natomiast w przypadku sieci o większej liczbie warstw mogą to być dowolne wielokąty (także nie wypukłe), a nawet obszary niejednospójne (złożone z wielu oddzielnych „wysp"). Myślę, że lepiej będzie to zobaczyć na ekranie, niż opisywać czy rysować, dlatego przygotowałem dla Ciebie program **Example 06d**.

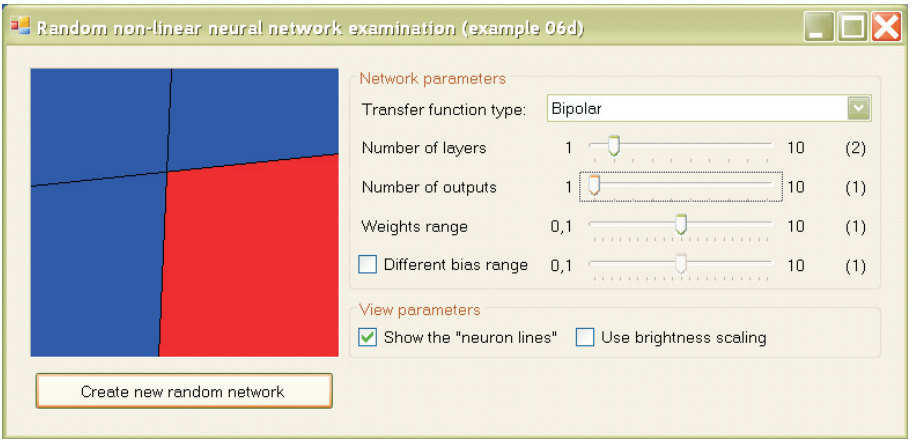



Rys. 6.12. Widok ekranu programu **Example 06c** przedstawiający mapę odpowiedzi neuronu

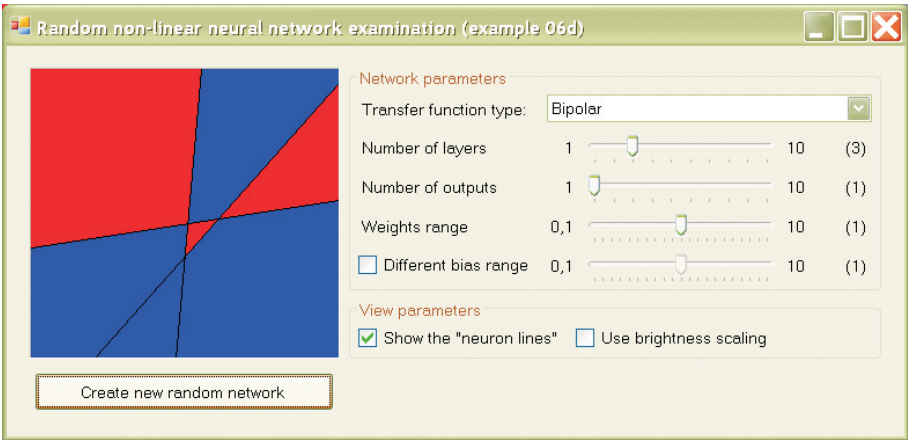
Uruchom go i pobaw się nim – pozwala on uzyskiwać podobne mapy, jak te, które budowałeś w poprzednim programie, ale szybciej i łatwiej, oraz – co najważniejsze – działa on dla sieci o dowolnej liczbie warstw. Jeśli wybierzesz sieć jednowarstwową – zobaczysz znany Ci już obraz dyskryminacji liniowej (rys. 6.12). Jeśli wybierzesz sieć dwuwarstwową – zobaczysz, że sieć zdoła w przestrzeni sygnałów wejściowych wyciąć podobszar spójny wypukły ograniczony płaszczyznami – tak zwany *simpleks* (rys. 6.13). Dla sieci mającej trzy (lub więcej) warstw – nie ma już żadnych ograniczeń. Obszar wyróżniany przez sieć w przestrzeni sygnałów wejściowych może być do-




Rys. 6.13. Przykładowy obszar reakcji sieci neuronowej o jednej warstwie




Rys. 6.14. Przykładowy obszar reakcji sieci neuronowej o dwóch warstwach




Rys. 6.15. Przykładowy obszar reakcji sieci neuronowej o trzech warstwach

wolnie skomplikowany, niekoniecznie wypukły i niekoniecznie spójny (rys. 6.14). Posługując się programem **Example 06d** możesz zobaczyć znacznie więcej różnych obszarów, ponieważ program losowo przyjmuje właściwości konkretnej sieci – możesz więc zobaczyć, jak rozmaite mogą być warianty opisanych wyżej kształtów obszarów. Ułatwią Ci to parametry którymi możesz zmieniać ustawienia sieci, m.in. dotyczące ilości warstw (**Number of layers**), funkcji aktywacji neuronu (**Transfer function type**), ilości wyjść sieci (**Number of outputs**). Program posiada wiele ciekawych funkcjonalności, których nie będziemy tutaj omawiać. Zachęcam Cię jednak, żebyś starał się odkryć te wszystkie możliwości „zaszyte” w rozważanym programie, a jeśli

jesteś zainteresowany programowaniem, to zachęcam Cię także do analizy jego kodu – jest to bowiem dobry przykład „sztuki programowania".

Oglądając rysunki produkowane przez ten program, możesz łatwo zauważyć, jak **ze wzrostem liczby warstw sieci wzrastają możliwości przetwarzania przez nią wejściowych informacji i możliwości realizacji złożonych algorytmów, wyrażających zależności pomiędzy sygnałami wejściowymi i wyjściowymi.** W omawianej wcześniej kategorii sieci liniowych takie zjawisko nie zachodziło: **sieć liniowa, niezależnie od liczby warstw, realizuje zawsze tylko i wyłącznie liniowe operacje przetwarzania danych wejściowych w wyjściowe,** praktycznie nie ma więc sensu budowa sieci liniowych wielowarstwowych. Przy sieciach nieliniowych – wprost przeciwnie. Tu zwiększenie liczby warstw jest jednym z podstawowych zabiegów stosowanych w celu uzyskania efektywniejszego działania sieci.

## 6.6. Jak przebiega uczenie nieliniowego neuronu?

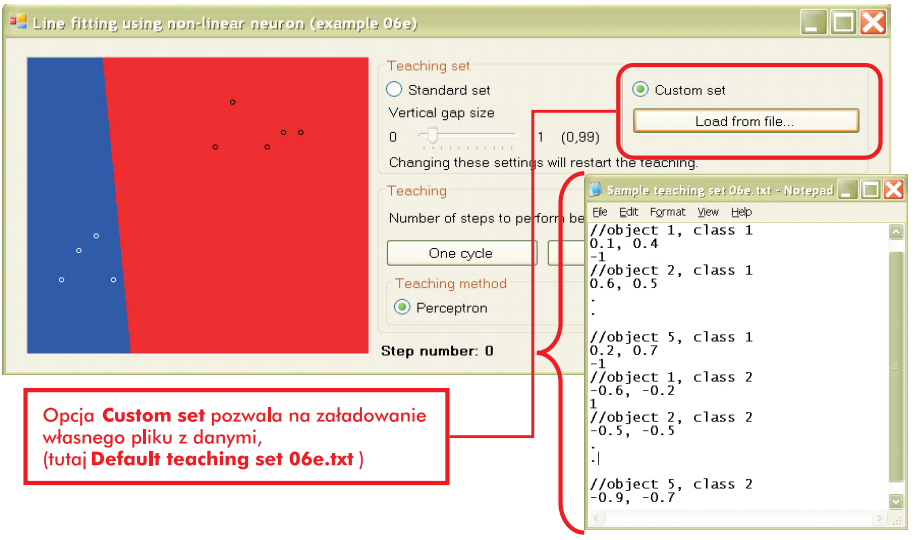
Przejdźmy teraz do zagadnienia interpretacji geometrycznej procesu uczenia sieci nieliniowej. Zaczniemy – jak zawsze – od najprostszego przypadku, tzn. procesu uczenia pojedynczego neuronu. Wiemy, że istota działania takiego neuronu polega na dzieleniu przestrzeni sygnałów wejściowych na dwa obszary za pomocą powierzchni liniowej (na płaskim rysunku będzie to po prostu linia prosta, ale w *n*-wymiarowej przestrzeni pojawiają się tu interesujące twory, zwane *rozmaitościami linowymi n-1 rzędu*, albo krócej – *hiperpłaszczyznami*). Ta powierzchnia musi być tak ustawiona, by możliwie dokładnie rozdzielała od siebie punkty przestrzeni wejść, dla których sieć powinna udzielać przeciwstawnych odpowiedzi.

Niżej przedstawię Ci kolejny program **(Example 06e)**, który pozwoli Ci prześledzić, jak przebiega proces uczenia nieliniowego neuronu i jak przebiegają zmiany ustawienia linii granicznej – na początku ustawionej w przypadkowym położeniu – aż do ulokowania jej (automatycznie!) w takim miejscu, by najlepiej rozgraniczała dwie grupy punktów, dla których w procesie uczenia podano przeciwstawne decyzje.

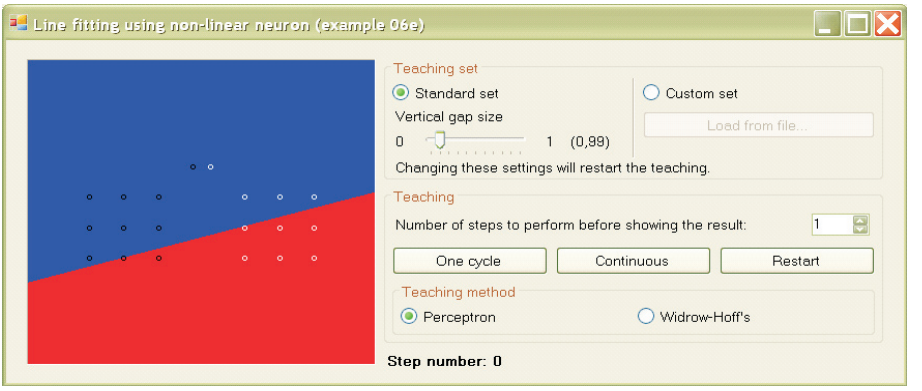
Program na początku pokazuje na ekranie dwa zbiory punktów (każdy z nich zawiera 10 obiektów), stanowiących ciąg uczący. Punkty należące do jednego zbioru przedstawione są jako małe, czarne okręgi, a do drugiego, jako podobne okręgi, ale białe. Zbiory te tworzą dwa wyraźnie oddzielne skupiska, jednak dla **utrudnienia** zadania w każdym zbiorze znajduje się dodatkowo pojedynczy „złośliwy" punkt wyraźnie wysunięty w kierunku przeciwstawnego zbioru. Wpasowanie linii granicznej – najpierw generalne pomiędzy zbiory, a potem precyzyjnie pomiędzy te „złośliwe", zbliżone do siebie punk-

ty – jest głównym zadaniem procesu uczenia. Położenia tych punktów możesz sam dowolnie zadawać, tworząc odpowiedni plik z danymi wejściowymi do programu **Example 06e,** a następnie wybierając opcję **Custom set** w grupie **Teaching set** w celu jego załadowania (rys. 6.16). Przykład takiego pliku znajdziesz w pliku „**Default teaching set 06e.txt**".

Na początek wygodnie jednak będzie „zapuścić" program z takimi parametrami, jakie są w nim ustawione (wybierając opcję **Standard set**). Przy opcji tej możliwa jest dodatkowo zmiana położenia zbiorów (**Vertical gap size**),




Rys. 6.16. Wybór i przykład własnego pliku z danymi wejściowymi do programu **Example 06e**



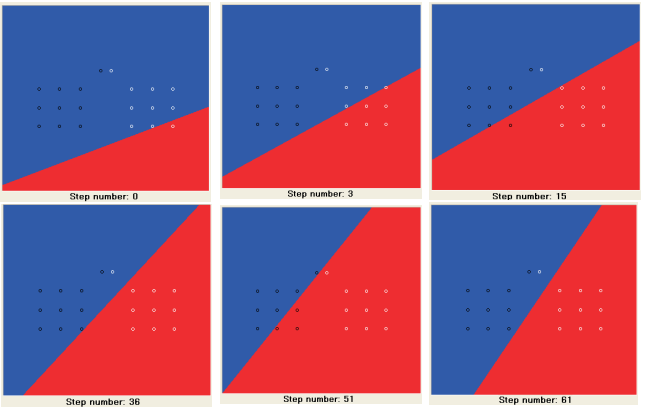

Rys. 6.17. Stan początkowy procesu uczenia w programie **Example 06e** po wybraniu opcji **Standard set** dla danych wejściowych

które możesz zbliżyć do siebie (zadanie neuronu staje się wtedy trudniejsze) albo odsunąć – zmieniając ich położenie za pomocą odpowiedzialnego za to „suwaka". Po uruchomieniu i wybraniu opcji **Standard set** zobaczysz taki obraz, jak na rysunku 6.17.

Na obrazie tym widoczne są wspomniane wyżej punkty oraz początkowe (przypadkowe, a więc za każdym razem inne) położenie linii granicznej (na styku dwóch półpłaszczyzn, zaznaczonych rożnymi kolorami). Oczywiście, początkowe położenie linii granicznej wynika z początkowych parametrów neuronu.

Jak zwykle w zadaniach uczenia, rolą neuronu w rozważanym zadaniu jest takie dopasowanie wag, by sygnały wysyłane na jego wyjściu były zgodne z zakwalifikowaniem punktów zbioru uczącego, podanym przez nauczyciela. Na ekranie widać będzie stale położenie obu zbiorów punktów, na których neuron jest trenowany, a także widać kolejne położenia linii granicznej – zmieniające się w trakcie uczenia. Początkowe położenie tej linii wynika z przypisania wszystkim wagom neuronu wartości przypadkowych. Taka początkowa „randomizacja" wag jest najbardziej rozsądnym rozwiązaniem, ponieważ nie wiedząc z góry, jakie wartości wag okażą się najwłaściwsze dla poprawnego rozwiązania – nie możemy zaproponować bardziej sensownego punktu startowego dla procesu uczenia – jak właśnie **losowy** zestaw wag. Jednak z tego powodu nie potrafimy też przewidzieć, jakie położenie przyjmie linia separująca rozdzielane punkty – bo jej początkowe usytuowanie (i wynikające z niego zmiany, jakim będzie podlegała w procesie uczenia) są właśnie wynikiem tego przypadkowo wybieranego punktu startu procesu uczenia.

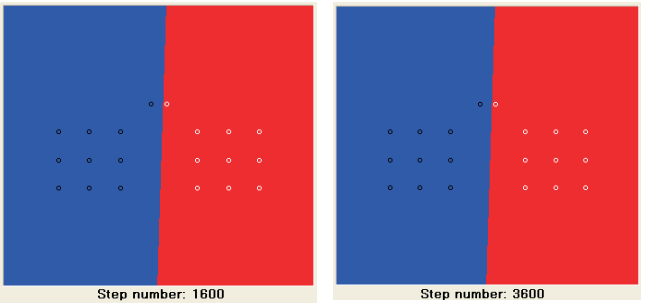
Obserwując, jak linia ta stopniowo zmienia położenie i „wciska się" pomiędzy rozgraniczane zbiory punktów, możesz się bardzo wiele dowiedzieć o naturze procesów uczenia w neuronach o nieliniowych charakterystykach. Uczenie z zadaną liczbą kroków w danym „cyklu" następuje po każdorazowym kliknięciu przycisku **One cycle**. W programie możesz zmienić liczbę kroków uczenia, które wykonywane są w trakcie jednego „cyklu", przed kolejnym wyświetleniem linii granicznej. Ma to spore znaczenie praktyczne, gdyż proces uczenia postępuje z początku szybko (kolejne pojedyncze kroki uczenia przynoszą łatwo zauważalne przesunięcia linii granicznej), ale potem dalszy postęp jest wolniejszy. Na początku warto więc podawać mniejsze liczby, np.: 1, 5 albo 10, natomiast potem trzeba już wykonywać „skoki" po 100 i po 500 kroków procesu uczenia pomiędzy kolejnymi pokazami (rys. 6.18). Abyś nie musiał nieustannie „klikać" w przycisk **One cycle**, przewidziano ułatwienie i po wciśnięciu przycisku **Continuous** program robi to za Ciebie aż do ponownego jego wciśnięcia. Program pokazuje za każdym razem, ile kroków uczenia łącznie wykonał (informacja **Step number:…** w dolnej czę-




Rys. 6.18. Zmiany położenia linii granicznej w początkowym etapie procesu uczenia neuronu

ści okna programu), więc będziesz miał na bieżąco kontrolę nad procesem uczenia w całej jego rozciągłości.

Spójrz na rysunek 6.18. Początkowe położenie linii granicznej przebiega w miejscu mocno odbiegającym od położenia docelowego, kiedy to – jak oczekujemy – po jednej stronie linii granicznej znajdą się wszystkie punkty należące do jednego zbioru, a po drugiej stronie znajdą się wszystkie punkty drugiego zbioru. Pierwszy etap uczenia (na rysunku 6.18 bardzo krótki – zaledwie 3 pokazy) spowodował pewien zauważalny postęp. Trend ten utrzymywany jest w kolejnych krokach – coraz więcej punktów jest dobrze rozgraniczanych. Niestety – krok 51 przyniósł pogorszenie uczenia i znowu więcej punktów znalazło się po złej stronie linii rozgraniczającej. Sieć podążając za „złośliwymi" punktami, które szczególnie trudno było rozdzielić, pogorszyła swoje działanie dla punktów, które należą do podstawowego zbioru łatwo rozróżnialnych danych. No cóż, takie kryzysy w procesie uczenia zdarzają się.




Rys. 6.19. Zmiany położenia linii granicznej w końcowym etapie procesu uczenia

Kolejne 10 kroków naprawiają sytuację, a następne etapy uczenia prowadzą do wyraźnej poprawy.

Jak widać na rysunku 6.19, proces uczenia zakończył się pełnym sukcesem, gdyż końcowa linia (na granicy czerwonej i niebieskiej półpłaszczyzny) separuje zbiory punktów idealnie dokładnie. Warto zauważyć, że po osiągnięciu prawidłowego rozwiązania dalsza kontynuacja uczenia już niczego nowego nie wnosi – na przykład linie pokazane po 1600 i np. po 3600 krokach uczenia pokrywają się. Tak jest zawsze, ponieważ po całkowitym nauczeniu neuron nie zmienia już swoich – poprawnych, jak wynika z sukcesu poprawnej klasyfikacji punktów – wartości współczynników wag.

## 6.7. Jakie badania możesz przeprowadzić podczas uczenia neuronu?

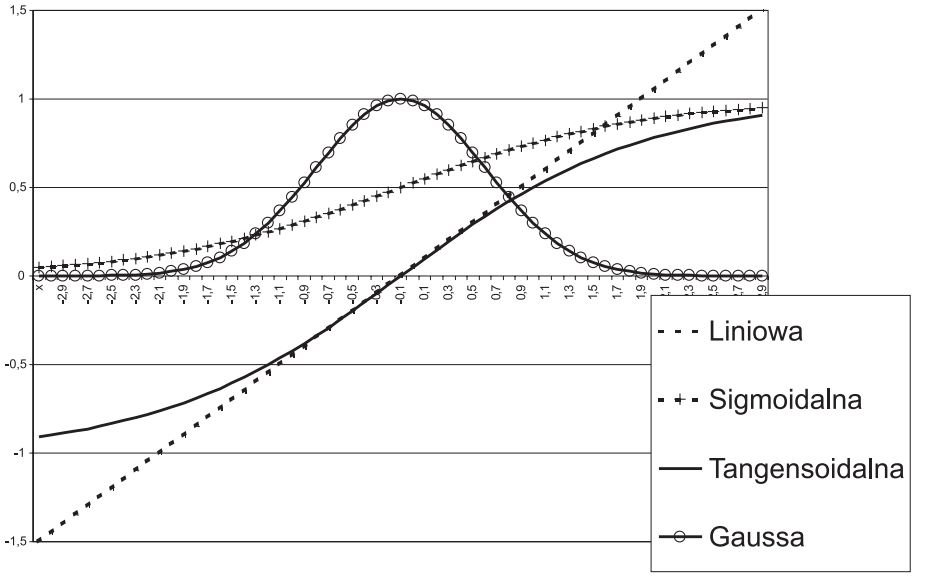
Opisany w poprzednim podrozdziale program **Example 06e** pozwala na samodzielne przeprowadzenie wielu różnych badań. Musisz tylko odpowiednio wybrać i ustawić interesujące Cię dane i parametry. Możesz w ten sposób zmieniać położenie poszczególnych punktów lub całych zbiorów uczących. Możesz również zastosować jedną z dwóch opisywanych w literaturze metod uczenia nieliniowego neuronu: albo perceptronową (podstawą uczenia jest wtedy sygnał wyjściowy z neuronu i jego konfrontacja z sygnałem wymaganym, podawanym przez „nauczyciela" (w rozważanym przypadku – przez ciąg uczący)), albo *Widrowa i Hoffa*. Ta druga metoda polega na tym, że uczeniu podlega wyłącznie część liniowa (podstawą uczenia jest „*net value*"). Wyboru pomiędzy tymi dwoma możliwościami dokonujesz w grupie **Teaching method**, oznaczając jedną z nich jako tę przez Ciebie aktualnie wybraną.

Prowadząc własne badania, zapewne zauważysz, że druga metoda uczenia, wiązana z nazwiskami *Widrowa i Hoffa*, bywa niekiedy bardziej skuteczna (być może już w pierwszych próbach uda Ci się odkryć, że rozgraniczenie zbliżone do idealnego zostaje znalezione w tej metodzie wyraźnie szybciej niż w metodzie perceptronowej), ale metoda ta nigdy nie przestaje się uczyć, więc po znalezieniu poprawnego rozwiązania tylko je psuje, potem naprawia, potem znowu psuje ...

Podobnie postępuje wielu ludzi, zauważyłeś? Między innymi znakomity polski malarz Aleksander Gierymski odznaczał się takim właśnie usposobieniem. Nigdy nie był do końca zadowolony ze swoich obrazów, więc po namalowaniu każdego kolejnego arcydzieła zaczynał poprawki – tu coś wytarł, tam coś domalował – i po pewnym czasie obraz nie nadawał się już do niczego. Wszystkie zachowane obrazy mistrza ostały się tylko dlatego, że jego przyjaciele siłą zabierali z jego pracowni „niedokończone" prace!

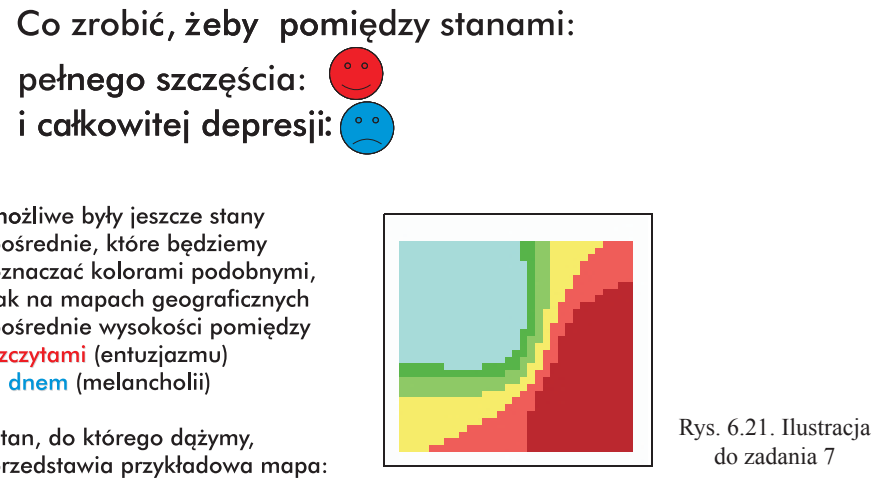
## 6.8. Pytania kontrolne, zadania i ćwiczenia do samodzielnego wykonania

1. Wymień przynajmniej trzy zalety, uzasadniające użycie **nieliniowych** neuronów przy rozwiązywaniu bardziej skomplikowanych zadań obliczeniowych.
2. Czy neuron **liniowy** ma charakterystykę unipolarną czy bipolarną?
3. Jakie zalety mają neurony z charakterystyką bipolarną? Czy występują one w mózgach ludzi i zwierząt? Dlaczego?
4. Na rysunku 6.20 pokazano najczęściej spotykane w praktyce kształty charakterystyk neuronów. Wskaż, które z nich są unipolarne, a które bipolarne.




Rys. 6.20. Najczęściej stosowane charakterystyki neuronów (ilustracja do zadania nr 4)

5. Na czym polega zjawisko uogólniania wiedzy, obserwowane w uczących się sieciach neuronowych?
6. Jakim ograniczeniom podlegają zadania, które mogą być rozwiązywane za pomocą – odpowiednio – jednowarstwowej, dwuwarstwowej i trójwarstwowej nieliniowej sieci neuronowej?
7. Co trzeba zmienić w definicji sieci neuronowej, żeby „zwierzak" opisywany w podrozdziale 6.4 mógł wykazywać stany pośrednie pomiędzy „pełnym szczęściem" a „całkowitym smutkiem", przedstawiane na przykład tak, jak na rysunku 6.19?




Rys. 6.21. Ilustracja do zadania 7

8. Naszkicuj wykres, pokazujący, jak zmienia się błąd popełniany przez sieć w trakcie procesu uczenia – na podstawie eksperymentów z klasyfikacją punktów należących do dwóch różnych klas. Jakie wnioski możesz wyciągnąć analizując kształt tego wykresu?

9. Dlaczego przy powtarzanych próbach uczenia **tej samej** sieci na bazie **tych samych** danych uczących uzyskuje się zwykle **różne** przebiegi procesu uczenia?

10. **Zadanie dla zaawansowanych:** Spróbuj uzasadnić twierdzenie (prawdziwe!), że w przypadku sieci liniowej sieć o wielu warstwach potrafi realizować tylko takie same proste odwzorowania, jak sieć jednowarstwowa, więc w sieciach liniowych wielowarstwowość nie przynosi żadnych korzyści.

11. **Zadanie dla zaawansowanych:** Zbuduj program, który pozwoli uczyć **wielowarstwową** sieć neuronową rozpoznawania klas złożonych z punktów **dowolnie** umieszczanych na ekranie poprzez klikanie myszką w odpowiednich punktach.

12. **Zadanie dla b a r d z o zaawansowanych:** Zbuduj program, który pozwoli sieci klasyfikować punkty w przestrzeni wejściowej w przypadku **pięciu**, a nie dwóch klas (jak to robiono w programie opisanym w tym rozdziale).

# 7. Backpropagation

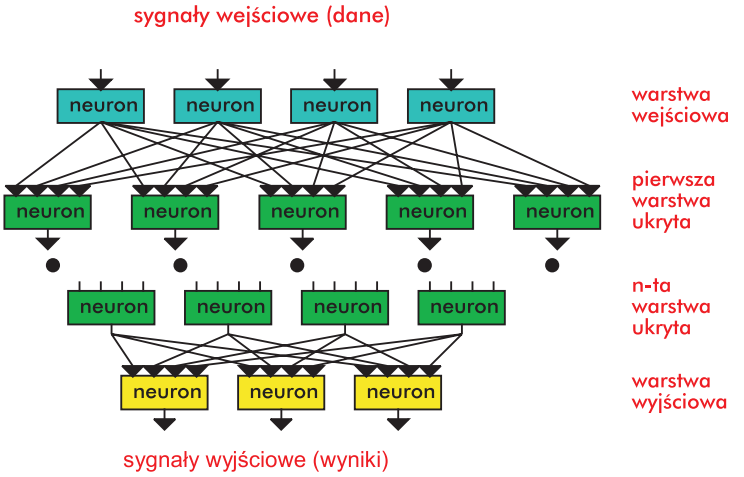
## 7.1. Co to jest backpropagation?

W poprzednim rozdziale rozważaliśmy wybrane aspekty funkcjonowania oraz uczenia **jednowarstwowej** sieci neuronowej zbudowanej z elementów nieliniowych. Tutaj będziemy kontynuować te rozważania, pokazując, jak mogą działać nieliniowe sieci **wielowarstwowe.** Sieci takie, jak już wiesz, mają istotnie większe i ciekawsze możliwości – co mogłeś sam stwierdzić, oglądając działanie programu **Example 06**. Dzisiaj porozmawiamy o tym, jak takie sieci można wykorzystywać i uczyć.

Wiesz już, że będziemy budowali sieci wielowarstwowe z nieliniowych neuronów. Wiesz już także o tym, jak można uczyć nieliniowy neuron (przypomnij sobie program **Example 06**)**.** Nie wiesz jednak (a nawet jeśli o tym słyszałeś, to nie poczułeś tego pewnie jeszcze na własnej skórze!), że problemem podstawowym przy uczeniu takich wielowarstwowych sieci neuronowych zbudowanych z nieliniowych neuronów jest problem tak zwanych **warstw ukrytych** (rys. 7.1)**.**

Na czym polega ten problem?

Otóż reguły uczenia, które poznałeś we wcześniejszych podrozdziałach, opierały się na prostej, ale bardzo skutecznej zasadzie: każdy neuron sieci sam wprowadzał poprawki do swego stanu wiedzy (zmieniając wartości współczynników wagowych na wszystkich swoich wejściach) **na podstawie znanej wartości błędu, jaki popełnił.** W przypadku sieci jednowarstwowej sytuacja była prosta i oczywista: sygnał wyjściowy każdego neuronu porównywany był z prawidłową wartością podaną przez nauczyciela, co dawało wystarczającą podstawę do korekty wag. W sieci wielowarstwowej nie jest już tak łatwo. Neurony końcowej (wyjściowej) warstwy mogą mieć oszacowane błędy w sposób w miarę prosty i pewny – jak poprzednio, poprzez porównanie sygnału produkowanego przez każdy neuron z wzorcowym sygnałem podawanym przez nauczyciela.




Rys. 7.1. Warstwy ukryte w wielowarstwowej sieci neuronowej

Natomiast neurony wcześniejszych warstw? Tu trzeba błędy oszacować matematycznie, bo wprost się ich zmierzyć nie da – brakuje wiadomości, jakie POWINNY być wartości odpowiednich sygnałów, bo nauczyciel nie określa tych wartości pośrednich, koncentrując się wyłącznie na efekcie końcowym.

Metodą, która powszechnie stosowana jest do „odgadywania" błędów neuronów warstw ukrytych, jest metoda zwana ***backpropagation*** (wstecznej propagacji błędów). Metoda ta tak bardzo jest popularna, że w większości gotowych programów służących do tworzenia modeli sieci i ich uczenia – stosuje się tę metodę jako domyślną, chociaż istnieje obecnie wiele innych metod uczenia, na przykład przyspieszona wersja tego algorytmu nazywana ***quick-propagation***, a także metody oparte na bardziej wyrafinowanych metodach matematycznych, takie jak ***metoda gradientów sprzężonych*** oraz ***metoda Levenberga–Marquardta***. Wymienione metody (a także przynajmniej tuzin innych, coraz bardziej wymyślnych) mają tę zaletę, że są bardzo szybkie. Jednak zaleta ujawnia się jedynie w przypadku, kiedy problem, który sieć neuronowa ma rozwiązać (dowiadując się sposobu tego rozwiązania na podstawie procesu uczenia), spełnia rozmaite wyrafinowane założenia matematyczne. Tyle tylko, że na ogół znając problem, który sieć ma rozwiązać, nie wiemy, czy problem ten spełnia owe skomplikowane założenia, czy też nie.

Co to w praktyce oznacza?

Otóż ni mniej ni więcej, tylko taką oto sytuację:

Mamy trudne zadanie do rozwiązania, bierzemy więc sieć neuronową i zaczynamy ją uczyć którąś z tych wytwornych i nowoczesnych metod – na przykład algorytmem Levenberga–Marquardta**.** Jeśli udało nam się mieć

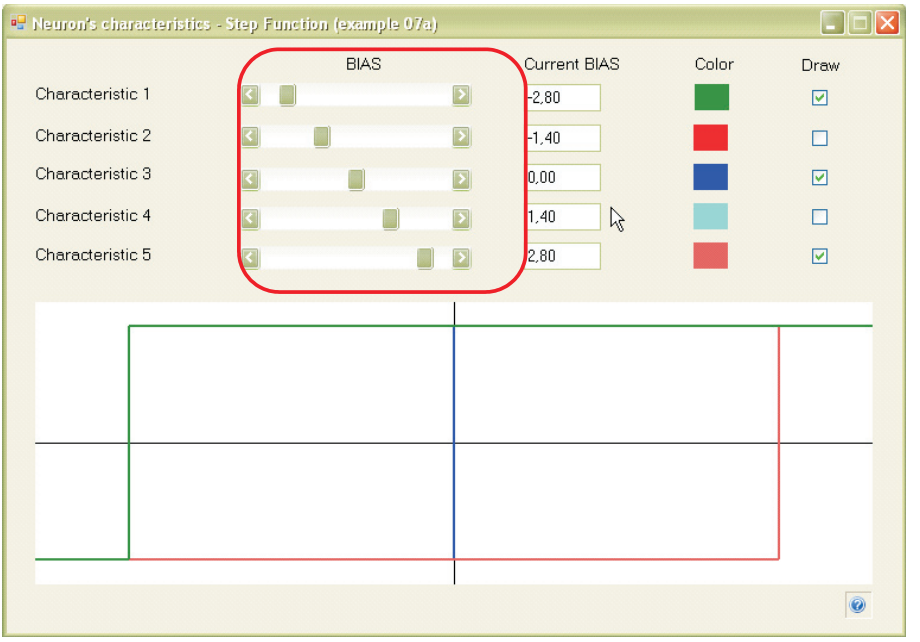
właśnie taki problem, jaki tym algorytmem łatwo się uczy – to sieć po bardzo krótkim czasie zostanie dobrze wytrenowana. Ale jeśli nie, to ten nowoczesny algorytm uczenia będzie nas bez końca wodził po manowcach, a sieć nigdy się niczego nie nauczy – bo nie są spełnione owe teoretyczne założenia. Tymczasem metoda ***backpropagation***, którą Ci w tym rozdziale przedstawię, ma jedną miłą cechę: otóż jej działanie jest niezależne od jakichkolwiek założeń teoretycznych. Oznacza to, że w odróżnieniu od tych różnych sprytnych algorytmów, które działają **czasami,** metoda ***backpropagation*** działa **zawsze.** Owszem, czasem może działać denerwująco wolno – ale nigdy nie zawiedzie. Warto więc ją poznać, bo ci, którzy profesjonalnie używają sieci neuronowych, często i chętnie do niej wracają – jak do wypróbowanego partnera.

Metodę ***backpropagation*** poznasz w działaniu, studiując zachowanie kolejnego programu, jaki za chwilę Ci zaproponuję. Zanim to jednak nastąpi musimy wrócić do jednego zagadnienia szczegółowego, które okaże się teraz bardzo ważne, a które do tej pory traktowane było trochę po macoszemu. Zajmę się mianowicie **kształtem nieliniowej charakterystyki** używanych w badaniach neuronów.

## 7.2. Jak zmieniać „próg" nieliniowej charakterystyki neuronu?

W poprzednim rozdziale rozważaliśmy neurony oparte na zasadzie „***wszystko albo nic***". Mogły one działać na zasadzie logicznej kategoryzacji wejściowych sygnałów (***prawda*** albo ***fałsz*** – **1** lub **0**) albo mogły opierać się na charakterystyce „bipolarnej" (***aprobata*** lub ***dezaprobata***, czyli sygnały **+1** lub **–1**). W obydwu rozważanych przypadkach przejście między dwoma wyróżnionymi stanami miało charakter gwałtowny: albo sygnały wyjściowe (sumarycznie) „przekraczały próg" i wtedy sygnał na wyjściu natychmiast przyjmował wartość **+1**, albo sygnały wyjściowe były słabe („pobudzenie podprogowe") – i wtedy odpowiedzią był brak reakcji (sygnał **0**) lub reakcja całkowicie negatywna (**–1**). Co więcej, przyjmowaliśmy zasadę zerowej wartości progu, to znaczy dodatniej sumie sygnałów wejściowych odpowiadał sygnał **+1**, a ujemnej **0** (lub **–1**) – co trochę ograniczało możliwości rozważanych sieci. Przy okazji omawiania kształtu funkcji przejścia nieliniowego neuronu warto więc także rozważyć zagadnienie wartości **progu**, gdyż w ogólnym przypadku wcale nie musi on być zerowy.

W rozważanych dziś nieliniowych modelach neuronów **próg** będzie uwolniony, co prowadzić będzie do uzyskania charakterystyk ruchomych, z możliwością dobierania punktu przełączenia w sposób całkowicie swobodny, za pomocą parametru zwanego zwykle w programach modelujących sieci jako **BIAS**. Przy pomocy programu **Example 07a** będziesz mógł narysować




Rys. 7.2. Rodzina progowych charakterystyk neuronu przy zmiennej wartości BIAS (widok z programu Example 07a )

sobie rodzinę progowych charakterystyk neuronów o zmieniającej się wartości BIAS – pozwoli Ci to na uzyskanie bardzo dobrego poglądu na temat roli tego czynnika w kształtowaniu zachowania się pojedynczych neuronów i całych sieci. Program ten pokazuje, jak wyglądać może zachowanie neuronu mającego swobodnie kształtowany próg (rys. 7.2).

## 7.3. Jaki jest najczęstszy kształt nieliniowej charakterystyki neuronu?

Charakterystyka rzeczywistego, biologicznego neuronu jest jednak jeszcze bardziej złożona. Pomiędzy stanem pełnego, maksymalnego (fizjolodzy mówią „tężcowego") pobudzenia a stanem podprogowym, wyrażającym się brakiem jakiejkolwiek aktywności, „rozpiętych" jest mnóstwo stanów pośrednich, wyrażających się występowaniem impulsacji o zmiennej częstotliwości. W uproszczeniu można powiedzieć, że im silniejszy jest bodziec wejściowy docierający do danego neuronu za pośrednictwem wszystkich jego wejść – tym większa jest częstotliwość impulsów na wyjściu neuronu. Można więc uznać, że w mózgu wszystkie informacje przekazywane są z wykorzystaniem

metody PCM[1], którą Przyroda wynalazła... o tych kilka miliardów lat wcześniej niż inżynierowie specjalizujący się w telekomunikacji!

Pełna dyskusja zagadnienia kodowania sygnałów neuronowych w rzeczywistym mózgu wykracza jednak poza zakres tej książki, zatem jeśli Cię to bliżej interesuje – zajrzyj może do mojej książki pt. *Problemy Biocybernetyki*, natomiast z punktu widzenia dalszej treści tego podrozdziału istotny jest tylko jeden wniosek z tych rozważań: możliwe (i celowe!) jest stosowanie sieci neuronowych zbudowanych z neuronów, na wyjściu których występować będą sygnały **zmieniające się w sposób ciągły** w zakresie od 0 do 1 lub od –1 do 1.

Takie neurony o ciągłych nieliniowych funkcjach przejścia odróżniają się korzystnie zarówno od neuronów **liniowych**, którymi zajmowaliśmy się dość długo we wcześniejszych rozdziałach, jak i od wprowadzonych w poprzednim rozdziale neuronów **nieliniowych nieciągłych,** mających tylko dwa dozwolone sygnały wyjściowe. Nie jestem w stanie przedstawić tu wszystkich tych zalet (zwłaszcza że część z nich daje się wypowiedzieć wyłącznie w formie pewnych twierdzeń matematycznych, a obiecałem, że w tej książce nie użyję ani jednego wzoru matematycznego), jednak nawet poglądowo możesz się łatwo zorientować, że nieliniowe neurony o ciągłych charakterystykach dają – stosunkowo – najszersze możliwości. Z jednej strony są to bowiem struktury **nieliniowe**, a więc mogą (w odróżnieniu od neuronów liniowych) formować wielowarstwowe sieci, umożliwiające ustalenie (w wyniku treningu) zupełnie **dowolnej zależności między wejściem i wyjściem.** Z drugiej jednak strony sygnały w tych sieciach mogą przyjmować w sposób płynny **dowolne wartości,** co pozwala na ich stosowanie w zadaniach, w których wynik obliczeń neurokomputera nie powinien się ograniczać wyłącznie do rozstrzygnięć typu „tak – nie", ale powinien określać jakąś wartość – na przykład spodziewaną zwyżkę kursu akcji w modnych ostatnio sieciach stosowanych do analizy rynku i przewidywań (predykcji) jego zmian.

Nieco mniej oczywisty, ale równie ważny jest drugi argument, przemawiający za rezygnacją z prostej, „skokowej" charakterystyki. Otóż dla skutecznego uczenia wielowarstwowej sieci neuronowej **KONIECZNE** jest, by budujące ją neurony miały charakterystyki **ciągłe i różniczkowalne[2].** Udowodnienie, że tak jest istotnie, jest stosunkowo łatwe, ale nie sądzę, by ta książka była najbardziej odpowiednim miejscem, aby taki dowód przytaczać.

[1] Jest to nowoczesna metoda impulsowego kodowania sygnałów, wykorzystywana w nowoczesnej elektronice, automatyce, robotyce, informatyce i telekomunikacji.

[2] Oba pojęcia – ciągłości i różniczkowalności są oczywiście pojęciami matematycznymi, a umawialiśmy się, że w tej książce matematyki nie ma. No, ale jeśli w tej chwili nie rozumiesz, co te pojęcia oznaczają, to ten fakt w najmniejszym stopniu nie utrudni Ci zrozumienia dalszego tekstu, bo ze wspomnianych właściwości absolutnie nie będziemy dalej (jawnie) korzystać.

Niemniej sam fakt warto zapamiętać – **funkcja przejścia neuronu w wielowarstwowej sieci uczącej się musi być dostatecznie „gładka”**, jeśli uczenie ma przebiegać bez zakłóceń.

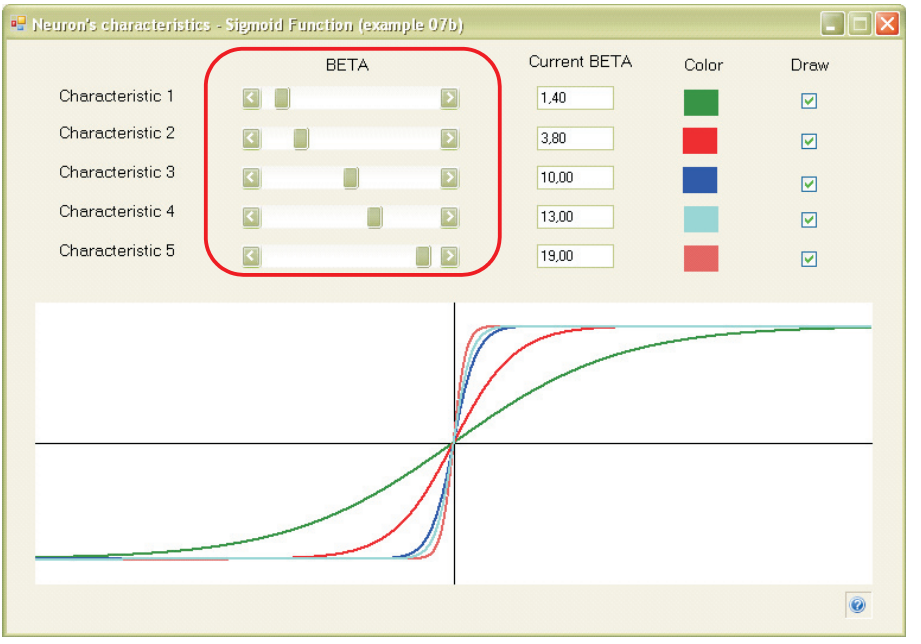
Można znaleźć wiele funkcji nadających się do tego, by mogły pełnić rolę funkcji przejścia dla modelowanych neuronów, największym powodzeniem cieszy się jednak **krzywa logistyczna,** zwana także (od kształtu jej wykresu) „***sigmoidą***”. Sigmoida ma następujące zalety:

⇒ zapewnia łagodne przejście między wartościami 0 a 1;

⇒ ma gładką i łatwą do praktycznego obliczenia pochodną;

⇒ ma jeden parametr (nazywany najczęściej „BETA”), którego wartość pozwala dowolnie wybierać kształt krzywej – od bardzo płaskiego, zbliżonego do funkcji liniowej, do bardzo stromego „przekaźnikowego” przejścia od 0 do 1.

Program **Example 07b** pozwoli Ci zapoznać się z tą funkcją nieco dokładniej. Łatwo zauważyć, że przy budowie nieliniowej funkcji przejścia neuronu uwzględnia się także możliwość przesuwania punktu „przełączenia” między wartościami 0 i 1 za pomocą parametru BIAS, ale dodatkowo możesz dobrać stromość krzywej i ogólny stopień nieliniowego zachowania neuronu (rys. 7.3).




Rys. 7.3. Różne formy krzywej logistycznej (widok z programu Example 07b)

Krzywa logistyczna, której właściwości poznałeś eksperymentując z przytoczonym wyżej programem, ma liczne zastosowania w naukach przyrodniczych. Kiedyś może napiszę o tym obszerniej, ale powiem tylko jedno: każdy **rozwój** (na przykład rozwój nowej firmy czy technologii) przebiega właśnie według krzywej logistycznej. Zastanów się przez chwilę, a przekonasz się, że tak jest w istocie: najpierw przyrosty są niewielkie i rozwój jest powolny. W miarę gromadzenia doświadczeń, kapitału i innych zasobów rozwój zaczyna przyspieszać i krzywa pnie się w górę coraz bardziej stromo. W chwili największego sukcesu, w najstromiej biegnącym w górę odcinku linii – rodzi się załamanie i krzywa ulega zagięciu w kierunku poziomym – na początku niezauważalnemu, potem jednak coraz bardziej radykalnie prowadzącemu do zatrzymania dalszego wzrostu krzywej, czyli „nasycenia". Końcem każdego rozwoju jest więc stagnacja, a po niej – czego już na krzywej logistycznej nie widać – nieuchronny upadek. No, ale to jest temat na zupełnie inne opowiadanie…

Odmianą sigmoidy dla sygnałów bipolarnych (symetrycznie rozłożonych między –1 i +1) jest **tangens hiperboliczny.** Nazwa brzmi groźnie, ale to naprawdę sympatyczna funkcja. Najlepiej obejrzyj ją sobie sam wykorzystując kolejny program **Example 07c**. Prawda, że w tej funkcji też nie ma niczego budzącego obawy?

Skoro już wiesz, jak wyglądają charakterystyki nieliniowych neuronów – pora zbudować z nich sieć i zacząć ją uczyć.


Rys. 7.4. Różne formy funkcji tangens hiperboliczny (widok z programu Example 07c).

## 7.4. Jak działa wielowarstwowa sieć złożona z nieliniowych elementów?

Sieci wielowarstwowe złożone z nieliniowych elementów stanowią obecnie podstawowe narzędzie służące do tego, by używać sieci neuronowych w praktycznych zastosowaniach. Służy do tego mnóstwo gotowych programów, z których możesz w razie potrzeby z powodzeniem skorzystać. O gotowych programach modelujących różnego typu sieci (zarówno ogólnie dostępnych – *public domain*, jak i bardzo licznych komercyjnych) opowiem jednak innym razem, tu natomiast spróbuję Ci przedstawić dwa proste programy, które unaocznią Ci najpierw, jak działa sieć wielowarstwowa (program **Example 08a**), a potem (co znajdzie się już w następnym podrozdziale) pokażę Ci program **Example 08b,** w którym poznasz sposób uczenia takiej sieci. Dzięki uważnemu przestudiowaniu tych dwóch podrozdziałów i dzięki eksperymentom, które wykonasz przy użyciu zaproponowanych programów, będziesz mógł sam zobaczyć, o co w tej słynnej metodzie *backpropagation* właściwie chodzi. Gorąco Cię namawiam, żebyś nie spieszył się i naprawdę uważnie przeczytał, przemyślał i dokładnie własnoręcznie wypróbował wszystko, o czym tu będzie mowa, gdyż ***backpropagation* jest solą i istotą techniki sieci neuronowych.** Jeśli poznasz i dokładnie zrozumiesz tę technikę – większość zagadnień związanych z sieciami będzie dla Ciebie łatwa i oczywista. Jeśli natomiast nie wciągniesz się i nie zaangażujesz w tę zabawę, którą Ci tutaj proponuję, to raczej nie będziesz tak do końca wiedział, *„co w trawie piszczy”*. Pewnie że można bez tego przeżyć, ale czy warto?

Zanim jednak uruchomisz te kolejne programy – przeczytaj kilka dodatkowych uwag na temat ich działania, które nie jest już (niestety!) tak oczywiste, jak działanie prostych programików, które pokazywałem Ci do tej pory.

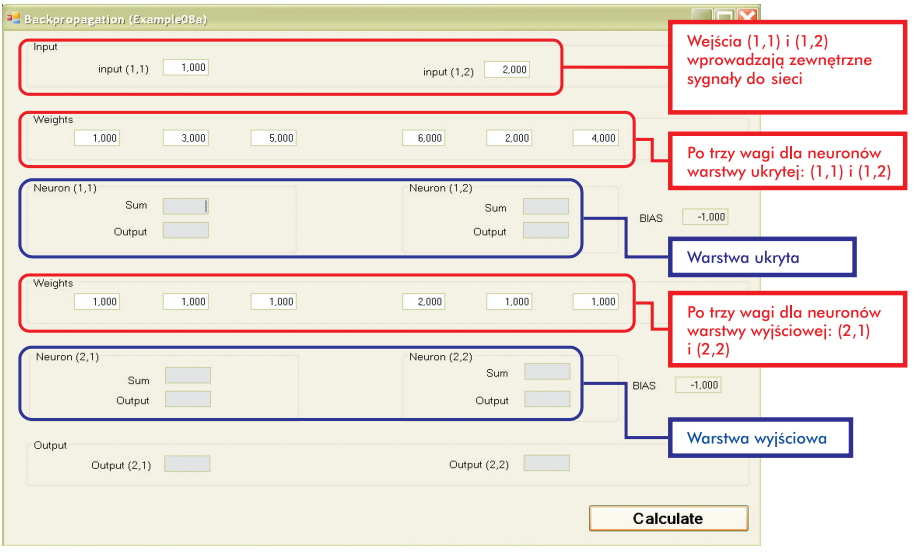
Program **Example 08a** pokazuje bardzo szczegółowo działanie sieci złożonej z czterech zaledwie neuronów (dzięki czemu łatwiej to będzie śledzić, patrz rys. 7.5) tworzących **trzy warstwy** – wejściową, nie podlegającą uczeniu (u góry ekranu), wyjściową, której sygnały będą kopiowane na wyjście sieci i tam będą podlegały ocenie i uczeniu (u dołu ekranu) i najważniejszą **warstwę ukrytą,** zaprezentowaną w centralnej części ekranu. Neurony i sygnały w sieci będą numerowane dwoma numerami: numerem warstwy i numerem kolejnym neuronu (albo sygnału) w warstwie.

Teraz kilka słów o działaniu programu i o jego obsłudze.

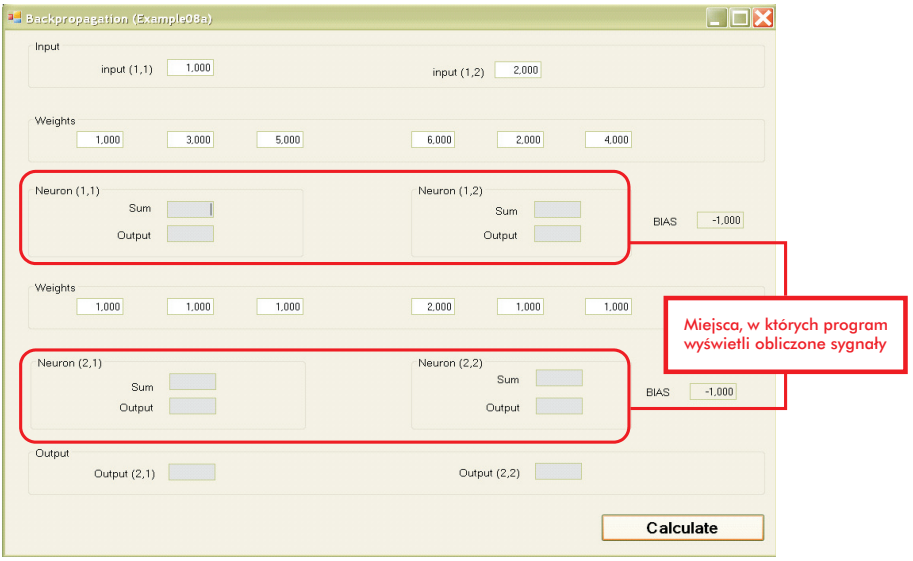
Na początku program prosi o podanie współczynników wagowych dla wszystkich neuronów w całej sieci (rys. 7.5). Współczynników tych musisz podać dokładnie 12, ponieważ – jak już wspomniałem – są 4 podlegające uczeniu neurony (dwa w warstwie ukrytej i dwa w warstwie wyjściowej),

a każdy z nich ma 3 wejścia (dwa od neuronów wcześniejszej warstwy plus dodatkowe wejście – też wyposażone we współczynnik wagowy – związane z „progiem", który reprezentuje w rozważanej sieci generator sztucznego, stałego sygnału BIAS). Trudząc się nad tym pomyśl, jak to dobrze, że w typowych sieciach, zawierających setki neuronów i tysiące współczynników wagowych, nie potrzeba wprowadzać ręcznie współczynników, tylko ustalają się one całkiem same, automatycznie – właśnie w trakcie procesu uczenia. W omawianym tutaj programie dałem Ci jednak możliwość „ręcznego" ustawiania wszystkich parametrów, ponieważ zależy mi na tym, żebyś mógł sprawdzić, czy wyniki, jakie sieć dostarcza na wyjściach swoich neuronów, są zgodne z tymi, jakie Ty sądzisz, że powinny być. W tym celu będziesz także mógł dowolnie kształtować sygnały wejściowe do sieci oraz obserwować sygnały zarówno na wejściach neuronów, jak i na ich wyjściach. Dla leniwych są oczywiście zaproponowane wartości domyślne… Najlepiej dobieraj proste liczby całkowite jako współczynniki wagowe i sygnały wejściowe, bo w ten sposób łatwo sprawdzisz, czy wyniki, jakie otrzymujesz z sieci, zgadzają się z tym, co sam obliczyłeś i sprawdzisz, czy naprawdę wiesz i rozumiesz, co się właściwie dzieje w sieci.

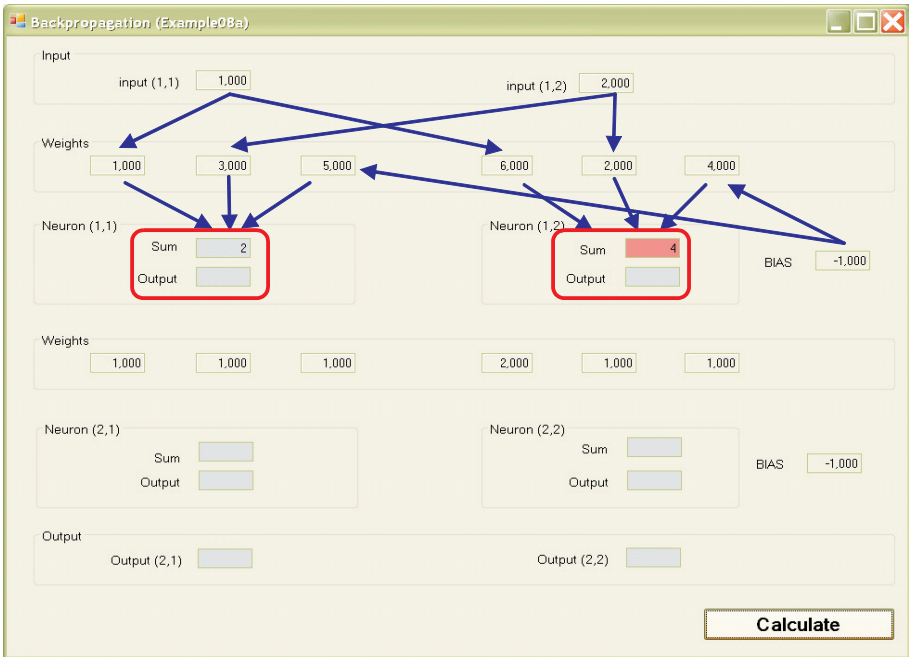
Po podaniu współczynników musisz dodatkowo podać wartości obydwu sygnałów wejściowych, jakie sieć ma otrzymać na swoich wejściach (rys. 7.5). W tym momencie program jest już gotowy do pracy. Na pewno zdążyłeś się już zorientować, jak przedstawiona jest struktura sieci (miejsca, gdzie




Rys. 7.5. Ustalanie parametrów i sygnałów wejściowych dla sieci w programie Example 08a




Rys. 7.6. Początek demonstracji sieci w programie Example 08a

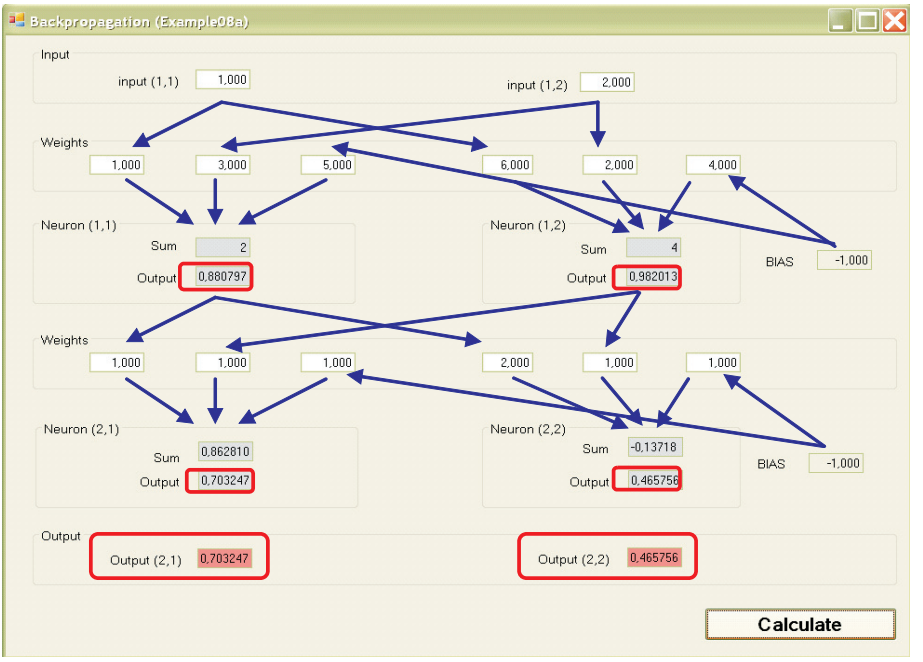



Rys. 7.7. Wizualizacja kierunku przesyłu sygnałów w początkowej części sieci

znajdują się poszczególne neurony oraz miejsca, gdzie wyświetlone będą poszczególne sygnały – patrz rys. 7.6).

Klikając teraz przycisk **Calculate** uruchomisz i będziesz prowadził proces symulacji procesu przesyłania i przetwarzania sygnałów w modelowanej sieci. Najpierw – w każdej kolejnej warstwie obliczane są i wyświetlane (na czerwonym tle) **sumy wartości sygnałów przemnażanych przez odpowiednie wagi** sygnałów wejściowych (wraz ze składnikiem BIAS). Następnie pokazywane są obliczone wartości **sygnałów wyjściowych** („odpowiedzi") poszczególnych neuronów (także początkowo na tle w kolorze czerwonym), powstałych po przepuszczeniu zsumowanych sygnałów wejściowych przez funkcję przejścia (o kształcie sigmoidy). Drogi przesyłania tych sygnałów oznaczone są na rysunku (rys. 7.7) w postaci niebieskich strzałek.

Następnie odpowiedzi neuronów niższej warstwy stają się wejściami dla neuronów wyższej warstwy i cały proces się powtarza (rys. 7.8).




Rys. 7.8. Końcowy etap działania programu – wszystkie sygnały zostały obliczone

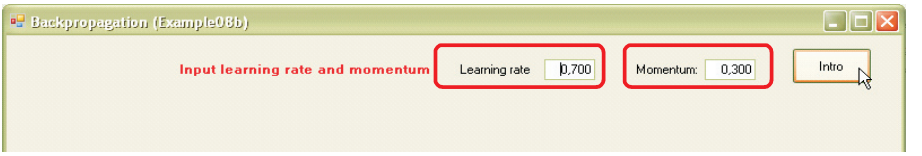
Naciskając kolejno przycisk **Calculate**, możesz obserwować przemieszczanie się sygnałów przez kolejne neurony od wejścia do wyjścia rozważanego modelu sieci. Możesz też zmienić wartości na wejściach i obserwować wyjścia – tyle razy, ile razy będziesz chciał sprawdzić, czy rzeczywiście już

dobrze rozumiesz, co się właściwie w tej sieci dzieje. Doradzam Ci, byś poświęcił trochę czasu na dokładne przeanalizowanie i przemyślenie każdej liczby na ekranie (może nawet sam sobie przelicz na boku na kalkulatorze, czy te liczby zgadzają się z tym, czego byś Ty się mógł spodziewać na podstawie Twojej wiedzy o sieciach neuronowych), bo tylko w ten sposób dowiesz się i porządnie zrozumiesz, co dokładnie robi sieć.

## 7.5. Jak można uczyć wielowarstwową sieć?

Gdy już opisana wyżej sieć wielowarstwowa, w której sam zadajesz wartości wag i wejściowe sygnały, przestanie mieć dla Ciebie jakiekolwiek sekrety – spróbuj wprowadzić i uruchomić tę samą sieć, ale jako **uczącą się**. Zrobi to dla Ciebie program **Example 08b.** Po uruchomieniu tego programu musisz podać wartości dwóch współczynników decydujących o przebiegu procesu uczenia. Pierwszy z nich określa wielkość poprawki, jaka wprowadzana jest po wykryciu każdego kolejnego błędu, a tym samym wyznacza całkowitą szybkość uczenia. Im większą wartość tego współczynnika podasz – tym silniej i szybciej zmieniać się będą wagi neuronów w wyniku wykrywanych w trakcie uczenia błędów. Współczynnik ten nazywa się w literaturze *learning rate,* więc ja go w programie nazwałem po prostu *współczynnik uczenia*. Współczynnik ten musi być wybierany z dużą rozwagą, gdyż zarówno zbyt duża, jak i zbyt mała wartość tego współczynnika fatalnie wpływa na przebieg procesu uczenia. Na szczęście nie musisz się zastanawiać, ile ten współczynnik powinien wynosić, gdyż znalazłem jego prawidłową wartość i program zaproponuje Ci tę właśnie wartość. Ale jeśli zechcesz – możesz zmienić podaną wartość współczynnika na dowolną inną i potem możesz się do woli dziwić, dlaczego wszystko „poszło w krzaki" – podobno żyjemy już w wolnym kraju i każdy może robić, co zechce!

Drugi współczynnik określa stopień „konserwatyzmu" procesu uczenia. W literaturze współczynnik ten określany jest jako *momentum*, co – jak łatwo możesz sprawdzić w słowniku – oznacza po prostu wielkość fizyczną, „pęd". Jak chcesz, to możesz go tak nazywać po polsku, ale potem nie dziw się, jeśli znajomi będą mówili o Tobie, że zajmujesz się podejrzanymi popędami w sieciach neuronowych. Ja pozostawiłem w programie oryginalną angielską nazwę. Im większa wartość współczynnika momentum, tym bardziej stabilny i bezpieczny jest proces uczenia – chociaż zbyt duża wartość może prowadzić do trudności z wykryciem przez sieć najlepszych (optymalnych) wartości współczynników wagowych dla wszystkich wchodzących w grę neuronów. Znowu znalazłem dla Ciebie stosunkowo dobrą wartość tego współczynnika. Wystarczy teraz, że zatwierdzisz obydwie zaproponowane wartości klikając




Rys. 7.9. Ustawianie parametrów wyznaczających sprawność uczenia w programie Example 08b

na przycisk **Intro** (patrz rys. 7.9). Możesz jednak próbować szczęścia i szukać lepszej jego wartości. Życzę powodzenia – to może się udać!

Po ustawieniu wybranych przez Ciebie (lub podanych przeze mnie) wartości współczynników, program prezentuje znane Ci już z wcześniejszych eksperymentów okienko, na którym widoczne są poszczególne neurony, wartości sygnałów, współczynniki wag i błędy. Dla ułatwienia w programie tym wprowadzono dodatkowy element wyjaśniający w postaci napisów wyświetlanych u góry ekranu, informujących Cię, co się w tym momencie dzieje. Te ułatwienia okażą się użyteczne i potrzebne, bo też zadania tego nowego programu będą bardziej złożone niż poprzedniego – będzie on bowiem uczył sieć.

## 7.6. Co obserwować podczas uczenia wielowarstwowej sieci?

Zadanie modelowanej sieci polega na tym, by prawidłowo rozpoznawać sygnały podawane na wejście. Idealne rozpoznanie polega na wystawieniu w wyjściowej warstwie sygnału **1** na neuronie przypisanym do danej klasy (odpowiednio lewym albo prawym); drugi neuron powinien oczywiście podawać w tym czasie wartość **0.** Do oceny poprawności działania sieci wystarczy jednak postawienie łagodniejszego warunku – wystarczy mianowicie, że błąd na żadnym z wyjść nie będzie większy[3] niż **0,5**, bo wtedy klasyfikacja sygnału wejściowego podana przez sieć nie różni się od klasyfikacji podanej przez nauczyciela. Jak z tego wynika – pojęcie **błędu** (mierzonego jako rozbieżność pomiędzy rzeczywistymi wartościami sygnałów na wyjściu sieci a wartościami podanymi przez nauczyciela jako wzorzec) nie jest tym samym, co **ocena poprawności rozwiązywania przez sieć postawionego zadania,** ponieważ

[3] Na przykład dla pojawiających się na pewnym etapie procesu uczenia sygnałów wejściowych wynoszących **–4,437 i 1,034** odpowiedzi sieci wynoszą **0,529 i 0,451**, co kwalifikowane jest jako błąd, ponieważ obiekt o podanych współrzędnych powinien być zaliczony do drugiej klasy, a tymczasem wyjście drugiego neuronu jest mniejsze niż pierwszego. Natomiast następny prezentowany przez program obiekt ciągu uczącego o współrzędnych **–3,720 i 4,937,** który wywołuje odpowiedzi **0,399 i 0,593**, może być uznany za rozpoznany poprawnie (jest to także obiekt drugiej klasy), chociaż wielkość średniokwadratowego błędu w obydwu przypadkach jest podobna (spróbuj zaobserwować podobną sytuację podczas swoich eksperymentów!).

to ostatnie wiąże się z generalnym zachowaniem sieci, a nie z wartościami konkretnych sygnałów. Można by zatem częściej sieci „odpuścić", jak jej się uda udzielić poprawnej odpowiedzi. Proces uczenia metodą *backpropagation*, jak zauważysz bawiąc się z opisanym tu programem, nie jest jednak skłonny do tolerancji. Jeśli na wyjściu miało być **1,** a neuron wyznaczył wartość **0,99** – to jest to błąd i trzeba go skorygować, nawet żeby uczeń płakał i fikał nogami! (przepraszam, poniosło mnie, fikania nogami jednak w programie nie przewidzieliśmy).

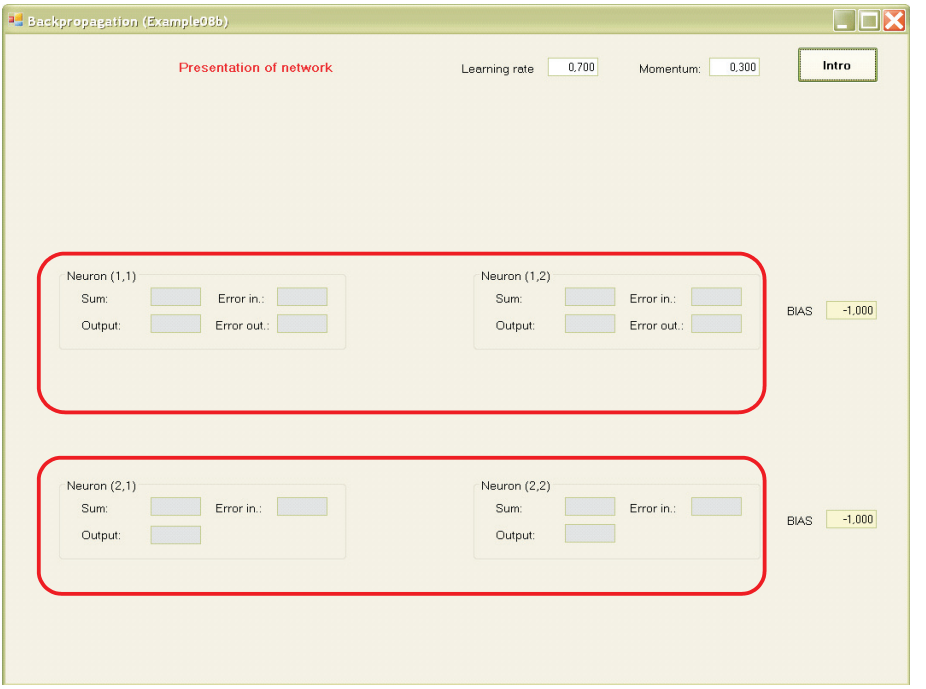
Taki perfekcjonizm, jak się okazuje, bardzo się opłaca, gdyż doskonaląc wartości swoich sygnałów zgodnie ze wskazówkami nauczyciela sieć uczy się przy okazji coraz skuteczniej osiągać także te praktyczne stawiane przed nią cele. Pojawia się tu bardzo ciekawa analogia z nauką w szkole – wprawdzie wiadomości nabywane w trakcie nauki – przykładowo – geometrii nie są bezpośrednio stosowalne w praktyce (ponieważ w rzeczywistości nigdy nie mamy do czynienia z idealnymi figurami geometrycznymi), jednak wiedza zdobyta dla tych idealnych tworów bywa przydatna do rozwiązywania praktycznych zagadnień, jakie niesie życie (na przykład w celu ustalenia, ile siatki potrzeba, by ogrodzić działkę o określonych wymiarach). Jeśli się ktoś nie przykłada do rozwiązywania pozornie bezsensownych teoretycznych zadań, jeśli przyswaja wiedzę podstawową tylko w zakresie „mniej więcej" – bywa potem w opałach, gdy zostanie skonfrontowany z prawdziwym, praktycznym zagadnieniem. Im lepiej nauczysz się rozwiązywać zadania matematyczne, tym skuteczniej będziesz sobie także radził z problemami praktyki. Nie dotyczy to ekonomii, gdyż dominujące obecnie w Polsce brutalne dążenie do tego, by za wszelką cenę maksymalizować zysk, tanio kupić i drogo sprzedać – oszukując przy tym, gdy się da, klienta – nie ma (jak twierdzi prof. Steczkowski z krakowskiej Akademii Ekonomicznej) żadnego związku ze szkolną matematyką i znajduje swój idealny wzorzec jedynie w łupieżczych wyprawach Wikingów. Natomiast jest sprawą bezsporną, że nauka wyidealizowanej fizyki pomaga w zrozumieniu bardzo praktycznej techniki, nauka biologii jest przydatna w uprawie roli i hodowli zwierząt, a nauka geografii pomaga w gospodarce przestrzennej. Z sieciami neuronowymi jest – jak widać – podobnie. Dążąc do osiągania **idealnych** celów stawianych im podczas nauki przez nauczyciela, zbliżają się one do wyznaczonej im roli **praktycznie** użytecznego narzędzia.

Wróćmy do omawianego programu. Podczas uczenia sieci Twoja rola ograniczać się będzie do obserwacji. Wystarczy więc, że będziesz siedział, naciskał dowolny klawisz i uważnie obserwował, co się dzieje na ekranie. Wszystko – początkowe wartości wag, sygnały wejściowe i wzorce poprawnych rozpoznań – program generuje sam. Niektóre z tych danych dla sieci

wybierane są w sposób losowy (początkowe wagi i sygnały wejściowe), ale wzorce poprawnych rozpoznań, na podstawie których sieć będzie się uczyć, nie są przypadkowe! Spróbuj jednak sam odczytać z programu lub na podstawie obserwacji jego działania, jaka zasada tu obowiązuje, bo nie chcę Ci psuć przyjemności. Zresztą nie musisz tej zasady znać – wystarczy, że sieć w trakcie procesu uczenia sama odkryje, o co tu chodzi – i będzie poprawnie klasyfikowała sygnały. Przekonasz się, że potrafi ona to zrobić, nawet wtedy, jeśli Ty – nie potrafisz!

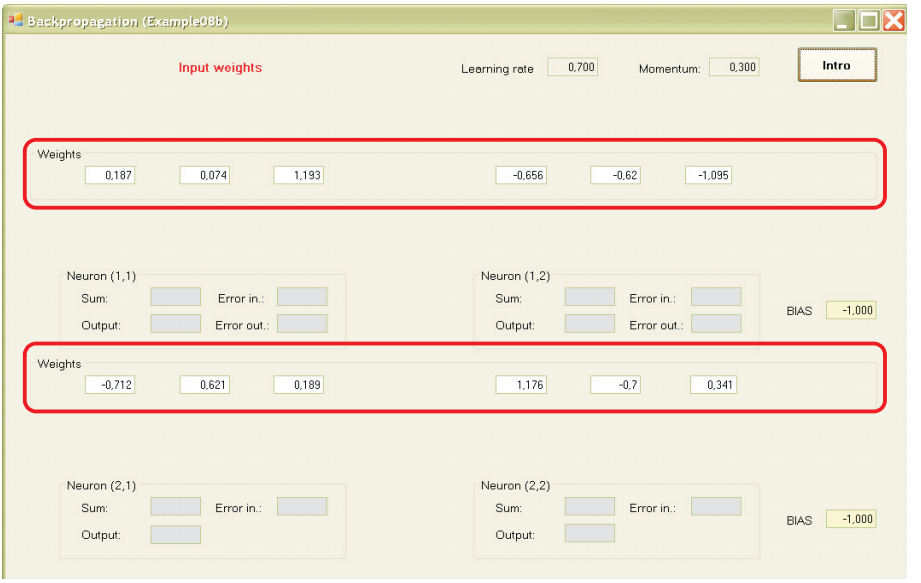
Teraz kilka słów o działaniu programu i o jego obsłudze.

Na początku program pokazuje tylko strukturę sieci, o czym zresztą usłużnie informuje napis u góry ekranu (w kolorze czerwonym) – rys. 7.10.




Rys. 7.10. Prezentacja struktury sieci

Zauważ, że źródła „pseudosygnałów" BIAS zaznaczone zostały tym razem na żółtym tle, żeby się nie myliły ze źródłami (i wartościami) prawdziwych sygnałów, uczestniczących w procesie uczenia. Potem, po kolejnym naciśnięciu **Intro**, program pokaże Ci **wartości współczynników wagowych** – wybrane na początku losowo dla wszystkich wejść wszystkich neuronów (rys. 7.11).

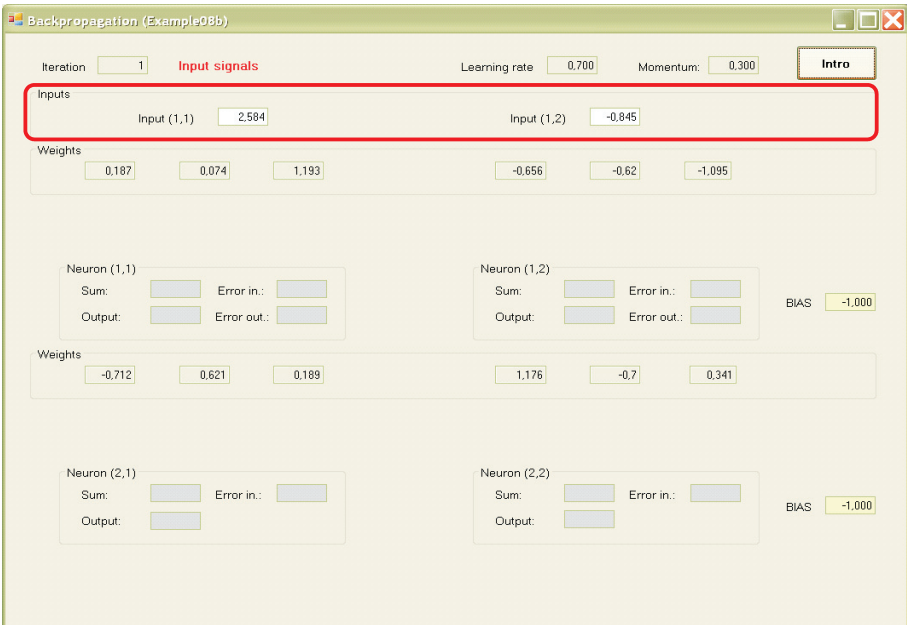



Rys. 7.11. Sieć i jej parametry

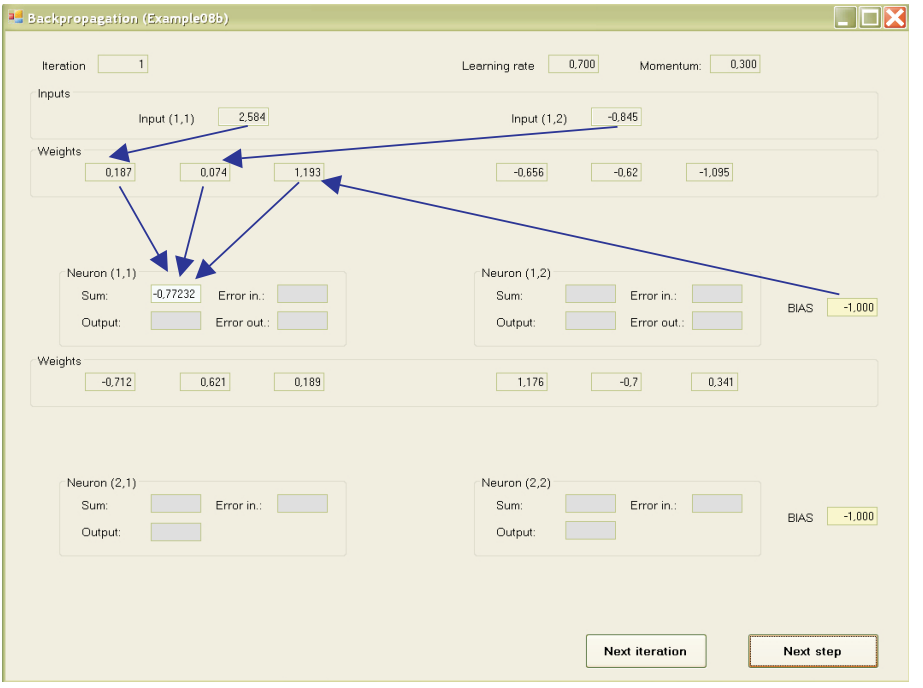
Gdy już sobie dobrze obejrzysz sieć i jej parametry – po kolejnym naciśnięciu klawisza **Intro** pojawiają się **wartości sygnałów wejściowych** podawane przez neurony oznaczone numerami (1,1) i (1,2) – rys. 7.12.

Od tego momentu zacznie się proces uczenia, jednak Twoja aktywna rola ograniczy się do tego, że kolejnym naciskaniem (klikaniem myszką) przycisku **Next step** uruchomisz i będziesz prowadził symulację modelowanej sieci (rys. 7.13).

Najpierw pokazane zostaną sygnały obliczone w poszczególnych neuronach podczas przesyłania sygnałów od wejścia do wyjścia. Jest to faza „propagacji w przód" – sygnały wejściowe przeliczane są przez neurony na ich sygnały wyjściowe i proces ten odbywa się kolejno na wszystkich warstwach – od wejścia do wyjścia. Tę fazę miałeś już okazję oglądać podczas eksperymentów z programem **Example 08a.** Obecnie używany program pozwala równie dokładnie śledzić przebieg tego procesu. Najpierw obliczane są i wyświetlane **sumy** wartości przemnażanych przez odpowiednie wagi sygnałów wejściowych (wraz ze składnikiem BIAS), a następnie pokazywane są obliczone wartości **sygnałów wyjściowych** („odpowiedzi") poszczególnych neuronów (tło jest wtedy podświetlane na czerwono), powstałych po przepuszczeniu zsumowanych sygnałów wejściowych przez funkcję przejścia o charakterystyce sigmoidy. Oczywiście odpowiedzi neuronów niższej warstwy stanowią wejścia dla neuronów wyższej warstwy, w związku z tym możesz

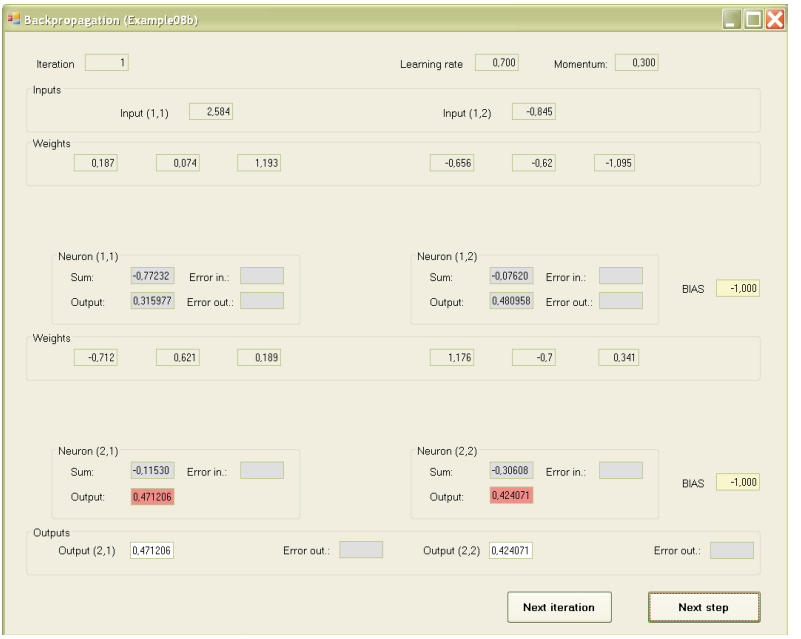



Rys. 7.12. Pojawienie się sygnałów wejściowych




Rys. 7.13. Wygląd ekranu podczas początkowego etapu symulacji działania sieci

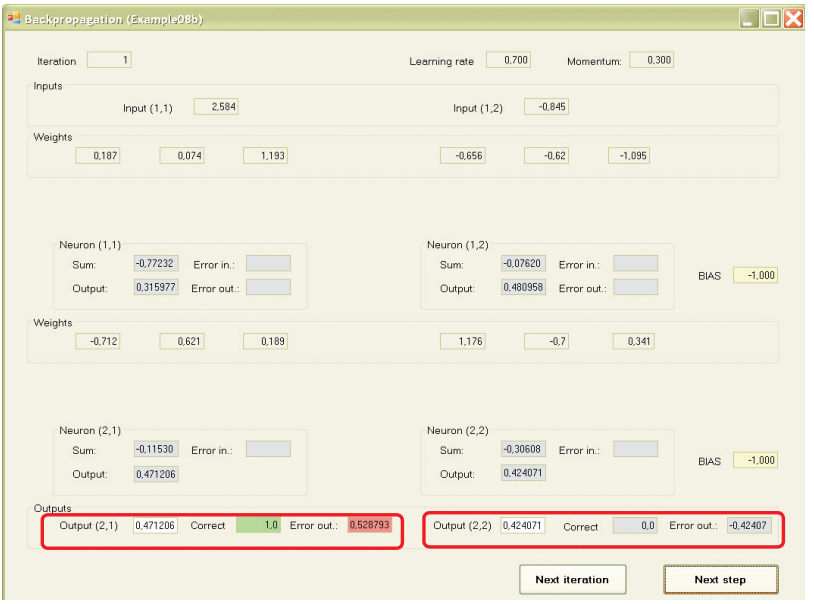
naciskać przycisk **Next step** i obserwować przemieszczanie sygnałów przez kolejne neurony od wejścia do wyjścia rozważanego modelu sieci. Docelowo, po wielu krokach wygląda to tak, jak na rysunku 7.14.



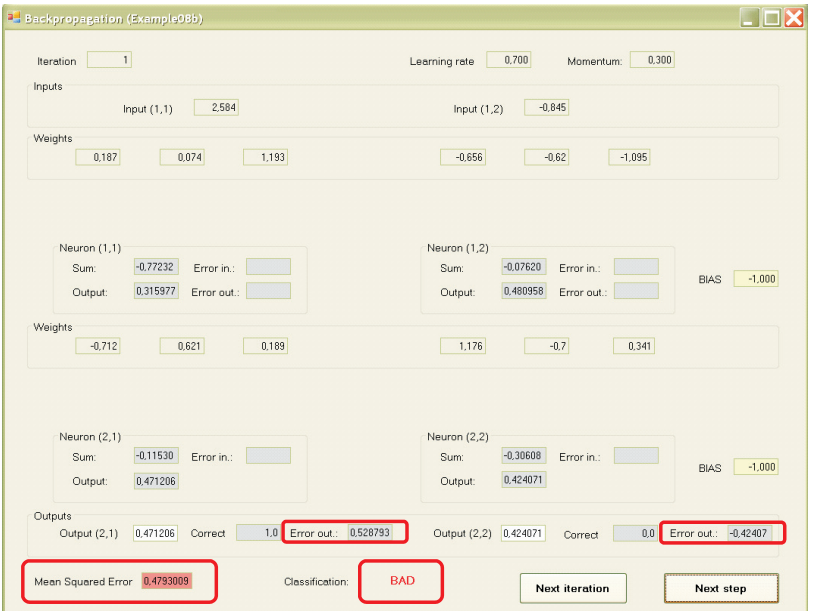

Rys. 7.14. Sieć po ukończeniu etapu propagacji w przód

Po ustaleniu się sygnałów wyjściowych na obydwu wyjściach sieci następuje (po kolejnym naciśnięciu przycisku **Next step**) „moment prawdy" – pokazywane są u dołu ekranu **sygnały zadane** (czyli wzorce poprawnej odpowiedzi) dla obydwu wyjść sieci (rys. 7.15), a następnie wyznaczane są **błędy** na wyjściach sieci oraz pokazany zostaje **błąd średniokwadratowy** obu wyjść. Na końcu wystawiana jest **końcowa ocena** dla całej sieci (rys. 7.16).

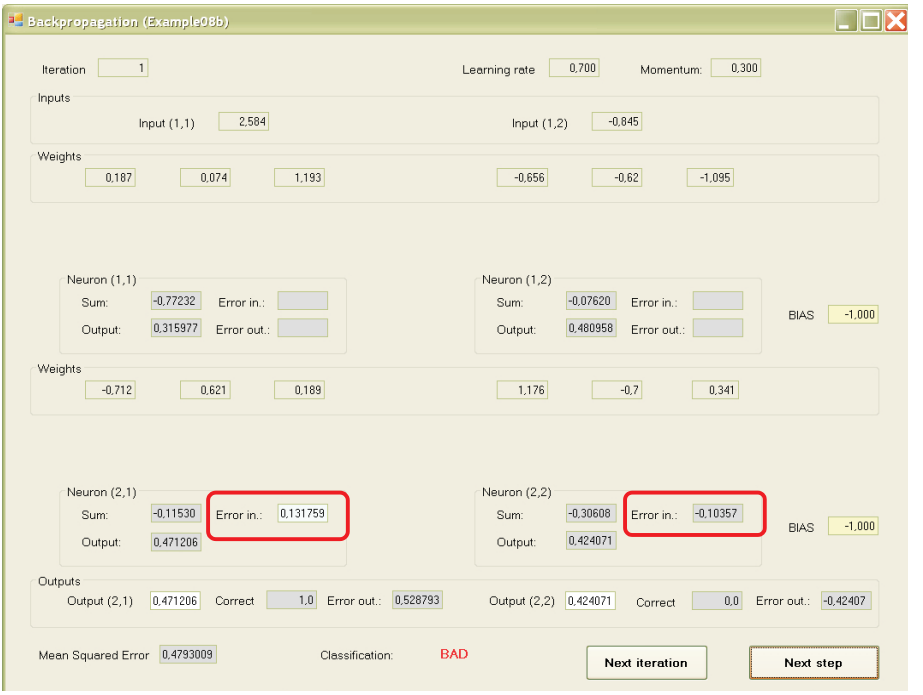
Na początku oceną (wystawioną – a jakże – na czerwono u dołu ekranu) będzie zapewne „**BAD**", wkrótce jednak sieć zacznie się uczyć i będzie osiągać pożądane oceny „**GOOD**". W momencie, gdy sieć (jako całość) popełni błąd – obliczane są błędy poszczególnych neuronów. Najpierw wyznaczane są wartości błędów dla wyjściowych neuronów sieci. Potem błędy te przeliczane są na odpowiadające im wartości na wejściach neuronów (do tego właśnie potrzebna jest różniczkowalna funkcja przejścia neuronu). Te **błędy przeniesione na wejście** zaznaczone są na rys. 7.17.




Rys. 7.15. Konfrontacja wyników działania sieci ze wzorcem podanym przez nauczyciela




Rys. 7.16. Wyznaczenia błędów neuronów warstwy wyjściowej i wypadkowego błędu całej sieci



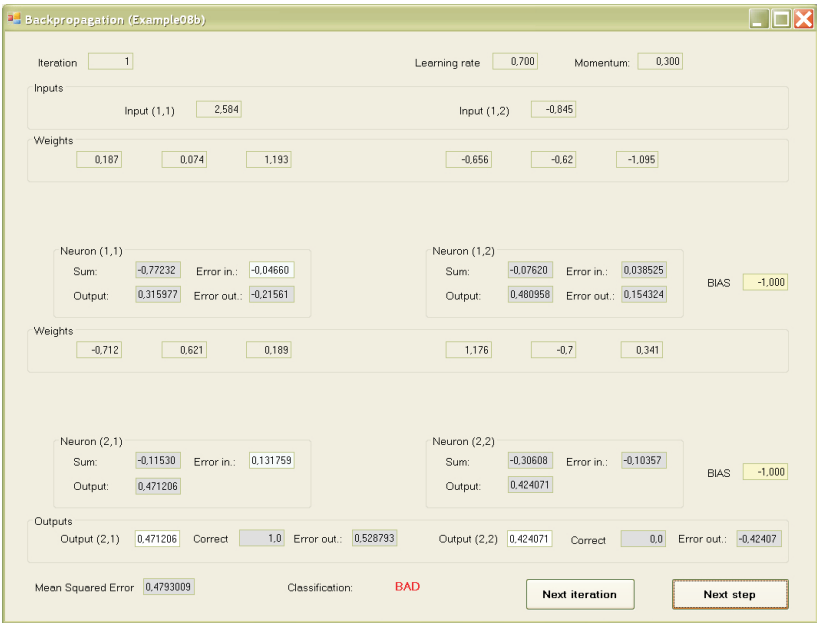

Rys. 7.17. Przeniesienie błędów na wejścia neuronów

Teraz następuje etap najważniejszy – ***propagacja błędów do neuronów niższej (ukrytej!) warstwy.*** Odpowiednie wartości pojawiają się przy odpowiednich neuronach – najpierw przy ich wyjściach, a potem przy wejściach – rys. 7.18.

Gdy już wszystkie neurony mają wyznaczone wartości błędów – kolejny krok programu prowadzi do wyznaczenia nowych (poprawionych) wartości współczynników wag w całej sieci. **Nowe współczynniki** pojawiają się poniżej dotychczas używanych współczynników, dzięki czemu możesz się dokładnie przyjrzeć, o ile i w jakim kierunku proces uczenia zmienił parametry sieci (rys. 7.19).

Jest to bardzo ważny moment w eksperymencie z programem. Wprawdzie na ekranie jest znaczna ilość informacji, ale **widać na nim wszystko** – sygnały wejściowe, wyjściowe, błędy, stare wartości wag, nowe wartości wag... Uważnie przeczytaj i przemyśl wszystkie informacje, które w tym momencie widzisz – im dokładniej zrozumiesz, jak to działa, tym więcej masz szans na przyszłe sukcesy w stosowaniu sieci.

Dalsze eksperymenty z programem możesz kontynuować na dwa sposoby. Jeśli uznasz, że chcesz jeszcze raz prześledzić cały przebieg kolejnej iteracji procesu uczenia: bieg sygnałów w górę, zwrotną propagację błędów w

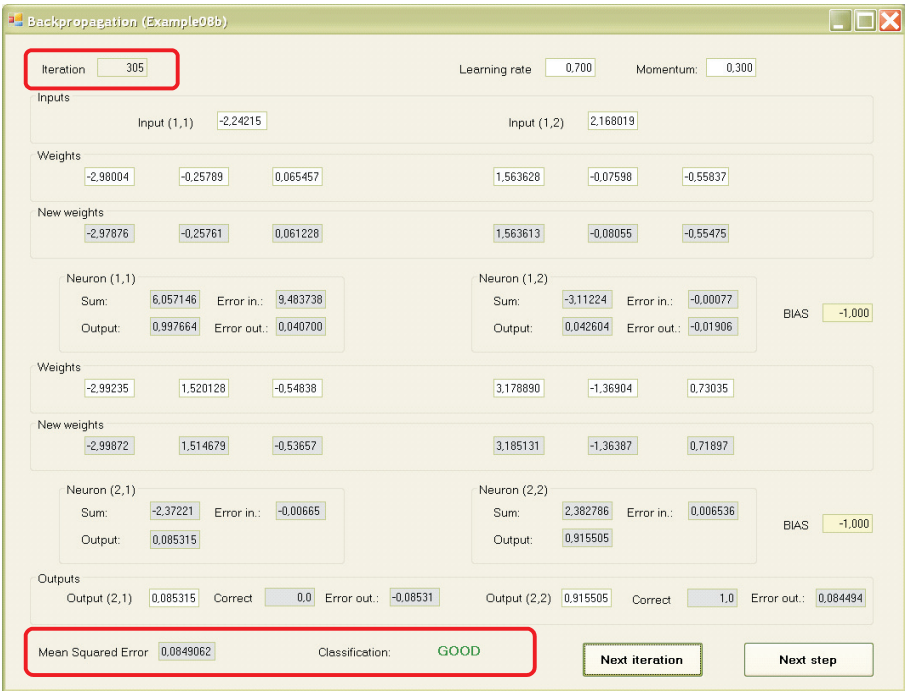



Rys. 7.18. Wynik wstecznej propagacji błędów


Rys. 7.19. Zmiana współczynników wagowych

dół, zmiany wag itd. – powinieneś naciskać **Next step**. Możesz również przyspieszyć cały proces poprzez wykonanie następnej kompletnej iteracji, czyli „skokowego" wykonania programu aż do momentu, kiedy ustalone zostaną nowe wartości wszystkich sygnałów, błędów i nowych wag. W tym celu użyj przycisku **Next iteration**.

Jest to najwygodniejszy na tym etapie tryb pracy, ponieważ takich pełnych iteracji procesu uczenia będziesz musiał wykonać bardzo dużo, zanim sieć zacznie się sensownie zachowywać. Dzięki temu możesz z pomocą tego programu łatwo i wygodnie obserwować postępy procesu uczenia, zmiany wag, a także to, co jest w omawianym tu algorytmie najważniejsze – proces rzutowania wstecz ustalanych w warstwie wyjściowej wartości błędów. Zobaczysz i wypróbujesz, jak wyznaczone dokładnie błędy neuronów wyjściowych i obliczone w procesie wstecznej propagacji błędy neuronów pozostałych („ukrytych") warstw pozwalają na znajdowanie wymaganych poprawek wartości współczynników wag dla wszystkich neuronów. Zobaczysz także, jak przez sukcesywne wprowadzanie poprawek posuwa się naprzód proces uczenia sieci. Radzę Ci to bardzo dokładnie prześledzić. Przykładowy efekt końcowy uczenia pokazuje rysunek 7.20.




Rys. 7.20. Efekt końcowy procesu uczenia

Zapoznanie się z programem **Example 08b** na pewno wymagało od Ciebie pewnego wysiłku, gdyż jest on stosunkowo duży i raczej skomplikowany. Uzyskałeś za to swoje własne (i łatwe do samodzielnej modyfikacji) narzędzie, pozwalające na dogłębne i gruntowne poznanie techniki *backpropagation* – jednej z najważniejszych metod uczenia sieci, wykorzystywanych we współczesnej technice sieci neuronowych. I uzyskałeś coś znacznie cenniejszego – wiedzę, rzetelną i dogłębną wiedzę na temat tego, co się właściwie dzieje w takiej sieci uczonej metodą *backpropagation.* Przekonasz się, jak bardzo ta wiedza się opłaci!

## 7.7. Pytania i zadania do samodzielnego rozwiązania

1. Dlaczego metoda *backpropagation* nosi właśnie taką nazwę? Co i dlaczego jest w niej „przesyłane wstecz”?
2. Co to jest BIAS i do czego służy? W jaki sposób zapewnia się możliwość uczenia tego parametru?
3. Dlaczego podczas pracy sieci w każdym neuronie wyświetlane są **dwie** wartości (**Sum** oraz **Output**)? W jaki sposób są one ze sobą powiązane?
4. Jaki wpływ na proces uczenia sieci mają odpowiednie współczynniki **Learning rate** oraz **Momentum?** Przypomnij sobie wiadomości teoretyczne na ten temat, a potem spróbuj zaobserwować wpływ tych parametrów podczas eksperymentów wykonywanych z programem.
5. Co oznacza proces przesyłania błędów z wyjścia neuronu (gdzie jest wyznaczany) na jego wejście (gdzie jest wykorzystywany do modyfikacji wag)?
6. Który z neuronów warstwy ukrytej będzie najsilniej obciążony błędem popełnionym przez określony neuron warstwy wyjściowej? Od czego to zależy?
7. Czy jest możliwa sytuacja, w której neurony warstwy wyjściowej wykazują błędy, a pewien neuron warstwy ukrytej ma przypisany (przez algorytm *backpropagation*) zerowy błąd i dlatego nie zmienia swoich współczynników wag?
8. Jak ustalana jest wypadkowa ocena działania całej sieci? Czy można jeszcze jakoś inaczej oceniać działanie całej sieci (a nie jej poszczególnych elementów, rozważanych indywidualnie)?
9. **Zadanie dla zaawansowanych:** Dodaj do programu moduł, który będzie pokazywał zmiany wartości błędów całej sieci oraz poszczególnych neuronów w postaci wykresów. Czy zawsze polepszenie działania całej sieci związane jest z minimalizacją błędów popełnianych przez wszystkie neurony sieci?

10. **Zadanie dla zaawansowanych:** Spróbuj rozbudować opisane w rozdziale programy w kierunku wykorzystania większej liczby sygnałów wejściowych i wyjściowych oraz większej liczby neuronów, formujących (jeśli chcesz szczególnie ambitnie potraktować to zadanie) więcej niż jedną warstwę ukrytą.

11. **Zadanie dla zaawansowanych:** Przy użyciu sieci korzystającej z większej liczby wejść spróbuj zaobserwować zjawisko uwalniania się procesu uczenia od nieprzydatnych informacji. W tym celu podawaj na jedno z dodatkowych wejść **bezużyteczne** informacje, nie mające żadnego związku z wymaganą odpowiedzią sieci (na przykład wartości z generatora liczb losowych). Na pozostałych wejściach muszą być podawane wszystkie informacje wymagane do tego, żeby sieć mogła rozwiązać postawione zadanie! Powinieneś zaobserwować, że po krótkim czasie uczenia wagi wszystkich połączeń prowadzących od takiego „pasożytniczego" wejścia do neuronów ukrytych przyjmą wartości bliskie zera, czyli to jałowe wejście zostanie praktycznie „amputowane"!

12. **Zadanie dla zaawansowanych:** Programy opisane w pracy prezentowały działanie sieci wyświetlając we wszystkich miejscach **wartości** odpowiednich parametrów i sygnałów. Taki sposób prezentacji pozwalał sprawdzić (na przykład przy użyciu kalkulatora), co i jak sieć robi, ale był w sumie mało czytelny w sensie obserwacji jakościowej zachodzących w sieci procesów. Zaprojektuj i wykonaj wersję programu, która wszystkie wartości przedstawia w formie graficznej, łatwiejszej i przyjemniejszej do interpretacji.

# 8. Formy uczenia sieci neuronowych

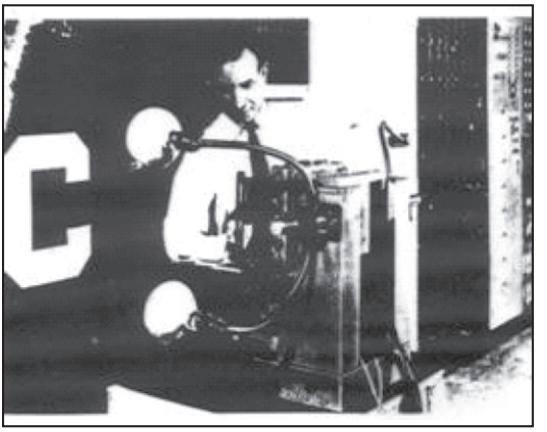
## 8.1. Jak wykorzystać wielowarstwową sieć neuronową do rozpoznawania?

Wielowarstwowe sieci neuronowe, które tak dogłębnie poznałeś w poprzednim rozdziale, mogą być używane do różnych celów. Wygodnie jest jednak rozważyć ich działanie i zachowanie w kontekście zadania tak zwanego „rozpoznawania obrazów". Rozpoznawanie obrazów jest zadaniem, w którym sieć neuronowa (lub inny automatyczny system rozpoznający) ma podejmować decyzje na temat przynależności określonych **obiektów** do ustalonych **klas.** Obiekty mogą być różnych rodzajów – mogą to być **obrazy cyfrowe** wprowadzone wprost z cyfrowych aparatów lub kamer cyfrowych, albo obrazy analogowe, przetworzone przez skaner lub „*frame grabber*". Porównanie obrazu analogowego i cyfrowego pokazuję na rysunku 8.1, żeby Ci lepiej


Rys. 8.1. Porównanie obrazu analogowego z obrazem cyfrowym

uświadomić, o czym właściwie tu mówimy, a także żeby zmniejszyć pewność siebie tych wszystkich, którzy są święcie przekonani, że „cyfrowy" to zawsze znaczy „lepszy".

Nie będziemy jednak w tej książce studiować dokładniej zagadnień rozpoznawania obrazów – zwłaszcza tych traktowanych w sposób dosłowny, bo książka ta dotyczy sieci neuronowych, a nie obrazów. Jednak warto przypomnieć, że od zadania rozpoznawania takich obiektów rozpoczął się rozwój tej dziedziny i od nich bierze ona swoją nazwę. Na rysunku 8.2 pokazano pierwszą (historyczną!) sieć neuronową, która została zbudowana przez Franka Rosenblatta i zajmowała się właśnie rozpoznawaniem obrazów (stąd jej nazwa: „Perceptron"). Zdjęcie to jest dosyć stare (z początku lat 70.) i dlatego niezbyt dobrej jakości – ale popatrz na nie z szacunkiem, bo to jest ślad jednego z najwcześniejszych osiągnięć w dziedzinie, o której jest mowa w tej książce.



Rys. 8.2. Widok „Perceptronu" Rosenblatta – pierwszej sieci neuronowej rozpoznającej obrazy

Pojęcie „obrazu" w problematyce rozpoznawania zostało potem uogólnione, tak że obecnie sieci neuronowe mogą również rozpoznawać **próbki sygnału dźwiękowego** (na przykład komendy wydawane za pomocą mowy), **sygnały sejsmiczne** (lub inne geofizyczne), pomagające w rozpoznawaniu złóż geologicznych, **symptomy pacjentów,** których choroby należy diagnozować, **oceny gospodarcze firm** ubiegających się o kredyty bankowe – i wiele innych. We wszystkich tych zadaniach mówi się o rozpoznawaniu obrazów, ale są to odpowiednio obrazy: dźwiękowe, geoinformatyczne, diagnostyczne, ekonomiczne i inne.

Sieć neuronowa rozpoznająca obrazy ma zwykle kilka **wejść**, na które podawane są sygnały odpowiadające wyróżnionym **cechom** rozpoznawanych obiektów. Mogą to być na przykład współczynniki opisujące kształt części maszyn oglądanych za pomocą kamery albo teksturę badanej ultrasonografem

tkanki wątroby. Na ogół takich wejść bywa dużo (na przykład w zadaniach rozpoznawania mowy sam używam sieci neuronowej o 98 wejściach), bo trzeba dokładnie „pokazać" sieci wszystkie cechy rozpoznawanego obiektu, żeby mogła się prawidłowo nauczyć go rozpoznawać. Jednak liczba cech, charakteryzujących obraz, jest znacząco mniejsza niż liczba elementów (pikseli) samego obrazu. Jeśli na wejście sieci rozpoznającej podasz bezpośrednio obraz cyfrowy, wówczas liczba wejść sieci (równa liczbie pikseli na takim obrazie) może sięgać setek tysięcy lub nawet wielu milionów! Dlatego praktycznie nigdy nie stosuje się takich sieci neuronowych, które pobierałyby wprost na swoje wejście analizowany obraz jako taki, tylko zwykle wprowadza się tam właśnie wspomniane **cechy** obrazu, wydobyte i ustalone za pomocą programów analizujących rozpoznawane obrazy **poza** siecią neuronową, co czasem bywa określane jako *preprocessing* tych obrazów.

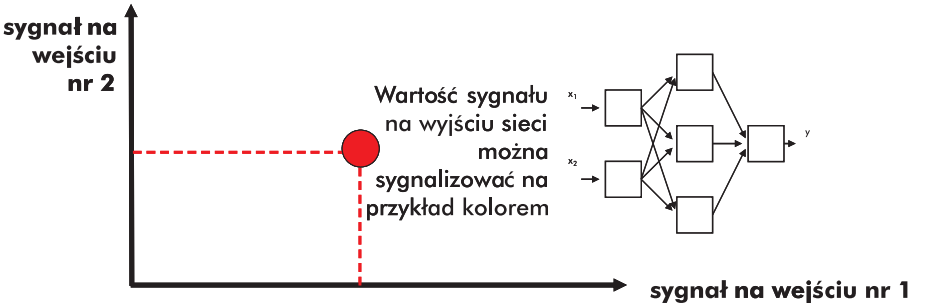
Sieć neuronowa rozpoznająca ma też zwykle wiele wyjść. Na ogół są one wykorzystywane w ten sposób, że do każdego wyjścia przypisuje się określone rozpoznanie. Na przykład w systemie automatycznego rozpoznawania znaków alfanumerycznych (tzw. zadania **OCR**) korzysta się z ponad 60 wyjść, z których każde przypisane jest do innego znaku – na przykład pierwszy neuron powinien sygnalizować pojawienie się litery **A**, drugi – **B** itd. Była o tym mowa w rozdziale 2, więc jeśli chcesz sobie przypomnieć, jak może wyglądać sygnał wyjściowy z sieci rozpoznającej obrazy – to spojrzyj może ponownie na rysunki 2.30, 2.31 i 2.33.

Pomiędzy wejściem i wyjściem jest też zwykle przynajmniej jedna warstwa ukryta – i tą właśnie warstwą ukrytą za chwilę nieco dokładniej się zajmiemy. Najogólniej mówiąc, warstwa ta może mieć więcej albo mniej neuronów – i potoczna interpretacja zjawisk zachodzących w sieciach neuronowych nakazuje nam uważać, że sieć mająca tam więcej neuronów jest bardziej „inteligentna" niż sieć mająca tych neuronów mniej. Jednak za chwilę przekonasz się także, że wcale nie jest dobrze dążyć do posiadania sieci o możliwie największej „wrodzonej inteligencji", gdyż sieć taka bywa czasem zaskakująco nieposłuszna!

## 8.2. Jak zaprogramowałem najprostszą sieć neuronową do rozpoznawania?

W przykładzie, który za chwilę wspólnie wypróbujemy, będziemy rozważali sieć o zaledwie **dwóch** wejściach. Prawdę powiedziawszy, w praktyce rzadko można coś ciekawego rozpoznać, biorąc za podstawę jedynie dwie cechy, bo sieci neuronowe chętnie się wykorzystuje głównie do analizy **wielowymiarowych** danych, czyli takich sygnałów wejściowych, które wymagałyby użycia

sieci o bardzo wielu wejściach. Jednak dla naszego przykładu założenie istnienia dwóch tylko wejść do sieci mieć będzie jedną, podstawową zaletę: **Każdy obiekt rozpoznawany przez sieć będzie można przedstawić jako punkt na płaszczyźnie.** Jedna współrzędna tego punktu odpowiada wartości pierwszej cechy rozpoznawanego obiektu, druga – wartości drugiej cechy (rys. 8.3).



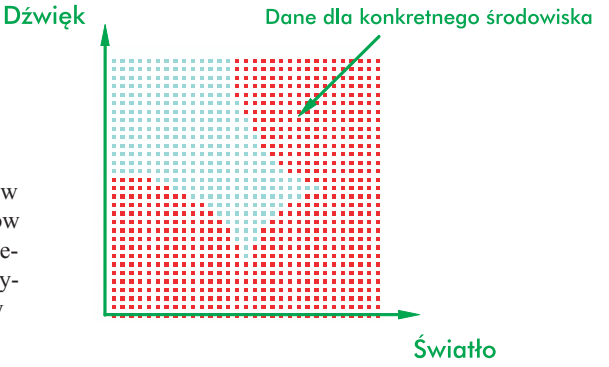

Rys. 8.3. Sposób reprezentacji sygnałów na wejściu i na wyjściu w przypadku sieci o dwóch wejściach i jednym wyjściu

Pamiętasz może, że rozważałeś już wcześniej podobną sytuację (w rozdziale 6.4)? Zaproponowałem Ci wtedy, abyś taką dwuwymiarową przestrzeń sygnałów wejściowych wyobraził sobie na przykład w ten sposób, że badana sieć jest mózgiem hipotetycznego zwierzęcia wyposażonego w dwa receptory – na przykład prymitywny wzrok i słuch. Im silniejszy jest sygnał odbierany przez wzrok – tym bardziej na prawo znajduje się punkt. Im silniejszy jest dźwięk – tym wyżej będzie się znajdował punkt na obrazku. Przypomnij sobie teraz tę analogię oraz związany z nią rysunek 6.8 – okaże się ona tutaj bardzo przydatna.

Dzięki takiej umowie każdy „pokazywany" sieci obraz (każde „środowisko", w jakim umieszczone będzie „zwierzę") będzie można wyświetlić na ekranie jako jeden punkt o ustalonych współrzędnych albo jako zapalony piksel ulokowany na ekranie w miejscu, którego współrzędne odpowiadać będą cechom rozważanego „środowiska". Oczywiście na to, by można to było skutecznie robić na ekranie o ograniczonych rozmiarach, trzeba będzie umówić się, że wartości rozważanych cech nie mogą być dowolne, lecz będą pochodziły z pewnego z góry ustalonego przedziału wartości. Konkretnie w rozważanym przykładzie widoczne na ekranie wartości obydwu współrzędnych (cech) stanowiących podstawę procesu rozpoznawania obrazów będą przyjmowane jedynie z przedziału od **–5** do **+5.** Fakt ten będziesz musiał brać pod uwagę formułując zadania, które sieć będzie rozwiązywać.

Każda sieć ma też wyjście, bo musisz jakoś obserwować jej zachowanie. W przedstawianym tu przykładzie zakładać będziemy zawsze, że jest to **jedno**

wyjście. Jego interpretacja też była już podana w podrozdziale 6.4 (na rysunku 6.8). Przypomnij sobie: Do każdej z pokazanych na wejściu sytuacji modelowane „zwierzę" może mieć nastawienie pozytywne (punkt wtedy będzie miał barwę intensywnie czerwoną) lub negatywne (punkt będzie miał kolor błękitny). Umieszczając nasze „zwierzę doświadczalne" kolejno w różnych środowiskach, czyli dostarczając do sieci neuronowej zestawy sygnałów wejściowych odpowiadające różnym lokalizacjom w przestrzeni sygnałów wejściowych (odbieranych przez receptory „zwierzęcia"), będziesz mógł sporządzić „mapę" pokazującą, gdzie Twój zwierzak dobrze się czuje, a gdzie nie. Mapa taka powinna być pokazana jako zbudowana z oddzielnych punktów, bo w takiej właśnie postaci pokaże Ci ją program, który za chwilę wspólnie uruchomimy (patrz rys 8.4), jednak dla wygody i estetyki dalszych rysunków będziemy rysować wszystkie mapy jako trochę „wygładzone" (patrz rys. 8.5).




Rys. 8.4. Przykładowa „mapa" pokazująca, do których warunków (tj. obszarów przestrzeni sygnałów wejściowych) modelowane „zwierzę" powinno mieć stosunek pozytywny, a do których negatywny

Rys. 8.5. Ta sama „mapa", co na rysunku 8.4, ale w postaci wygładzonej

Możliwe też będą (w miarę komplikacji sieci – coraz częstsze) stany pośrednie, częściowa niechęć (kolor jasnoniebieski), nastawienie obojętne (kolor jasnozielony) i częściowa aprobata (kolor żółty, przechodzący potem w jasnoczerwony) – rys 8.6. Jak na mapie w Twoim atlasie geograficznym – od błękitnych głębin smutku do ciemnej czerwieni szczytowego entuzjazmu!

Rys. 8.6. Przykładowa mapa „preferencji" badanego „zwierzęcia" na pewnym etapie procesu nauki

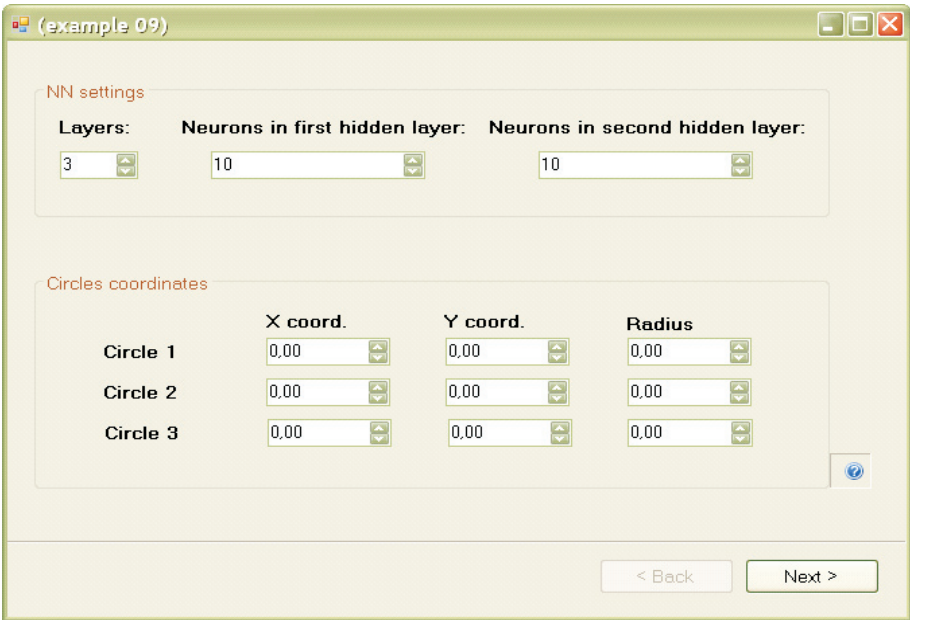
Dzięki temu, że sieć ma tylko jedno wyjście, będzie można także łatwo i wygodnie pokazać na ekranie, jak ta sieć **powinna** działać. Zapalając na ekranie punkt odpowiadający określonemu zestawowi współrzędnych wejściowych będziesz mógł zobaczyć, czy punkt ten należy do klasy „akceptowanej" (wówczas zapalany będzie w kolorze czerwonym) czy „odrzucanej" (w takim wypadku zapalany będzie na ekranie punkt w kolorze błękitnym). Oczywiście, dane opisujące te punkty trzeba będzie odpowiednio podać w programie, ale o tym powiem Ci nieco dalej.

Takie mapy będziesz oglądał w toku procesu uczenia sieci. Oczywiście będą się one zmieniały, bo w wyniku uczenia sieci będzie dochodziło do tego, że sieć przestanie „lubić" coś, co wcześniej wręcz „uwielbiała", albo „polubi" jakieś warunki, które na początku wywoływały jej zdecydowaną odrazę. Przekonasz się, że ogromnie ciekawe będzie śledzenie, jak początkowa niepewność i niezdecydowanie sieci w miarę postępu procesu uczenia przekształcać się będzie w „hipotezy robocze", które potem krystalizować się będą w „absolutną pewność". Czegoś takiego nie dałoby się równie prosto i sugestywnie pokazać dla sieci o wielu wyjściach, chociaż, jak już wspomniałem, wyjść w rzeczywistych systemach rozpoznawania bywa **z reguły** więcej niż jedno.

Przyjęcie w prezentowanym niżej przykładzie tylko jednego wyjścia ma także tę dodatkową zaletę, że nie trzeba się będzie zastanawiać, czy sieć, która – przykładowo – z dużym zdecydowaniem wskazuje na kilka potencjalnie możliwych zaklasyfikowań sygnałów wyjściowych (wśród których jest to właściwe), jest lepsza (czy może gorsza) od sieci, która wszystkie odpowiedzi podaje jako bardzo słabe i niepewne – chociaż wśród tych małych wartości – zdecydowanie największa przypada na wyjście tego neuronu, który powinien sygnalizować prawidłowo rozpoznany obiekt. Takich problemów z sieciami o wielu wyjściach może być więcej i dlatego w sieciach takich bardzo trudne są oceny aktualnego działania sieci, jego zmian w trakcie procesu uczenia, ewentualnych reakcji na sytuacje nietypowe (na przykład zdolność sieci do uogólniania wiedzy) itp. W sieci o jednym wyjściu sytuacja jest prymitywna, ale zawsze klarowna i pewna: albo rozpoznanie jest poprawne, albo nie.

## 8.3. Jak wybierać strukturę sieci neuronowej w trakcie eksperymentów?

Program **Example 09** umożliwi Ci eksperymenty z siecią rozpoznającą. Na początku pracy programu wyświetlony zostanie panel, w którym będziesz mógł wprowadzić parametry opisujące strukturę sieci, którą chcesz stworzyć i przebadać (rys. 8.7). Jak zwykle zostaną Ci zaproponowane pewne domyślne parametry, które jednak możesz modyfikować, i tak właśnie na początku naszych eksperymentów zrobimy.




Rys. 8.7. Przykład ustalania struktury sieci w programie **Example 09**

Ustaliliśmy, że sieć, którą będziesz badać i uczyć, ma strukturę:

**2 – xxx – 1**,

gdzie kolejne liczby oznaczają liczbę neuronów w kolejnych warstwach: dwa na wejściu, jeden na wyjściu i pewna liczba (**xxx** – do ustalenia) w środku. Wybierając konkretną sieć, będziesz musiał tylko podjąć decyzję, ile tych wewnętrznych neuronów ma być i jak mają być uporządkowane (to znaczy ile „ukrytych" warstw mają tworzyć).

W zależności od tego, jaką liczbę warstw wprowadzisz w polu o nazwie **Layers**, uzyskasz możliwość ustalenia liczby neuronów w poszczególnych warstwach. Wybierając liczbę warstw pamiętaj, że brane są pod uwagę tyl-

ko te, w których znajdują się neurony podlegające uczeniu – a więc wliczana jest jednoelementowa „warstwa" wyjściowa, natomiast **nie są wliczane** dwa neurony warstwy wejściowej, które jedynie przyjmują i „rozprowadzają" wejściowe sygnały. Jeśli zatem podasz, że chcesz mieć sieć jednowarstwową – otrzymasz sieć o strukturze:

**2 – 1**

i nie musisz ustalać liczby neuronów, bo z założenia wiadomo, jaka jest. Jeśli zażądasz sieci dwuwarstwowej, będziesz miał strukturę:

**2 – x – 1.**

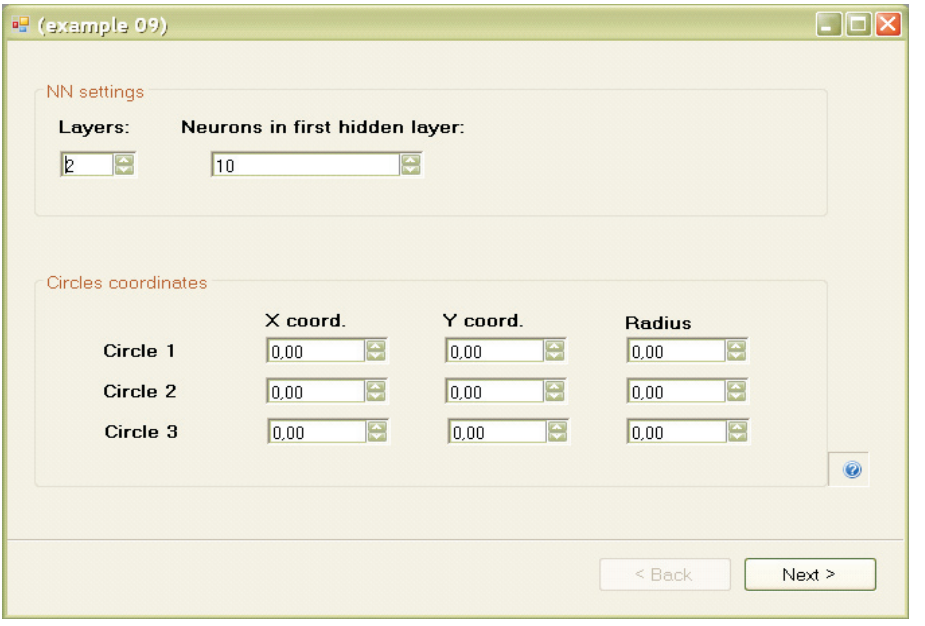
W tym przypadku ukaże się automatycznie pole w którym będziesz mógł podać, ile neuronów warstwy ukrytej ma Twoja sieć (**Neurons in first hidden layer**) – rys. 8.8.

Wreszcie gdy podasz, że sieć ma mieć trzy warstwy (lub pozostawisz początkowe, domyślne ustawienia) będziesz miał strukturę:

**2 – x – y – 1**

i będziesz musiał podać liczbę neuronów w obydwu warstwach ukrytych (w polach **Neurons in first hidden layer** i **Neurons in second hidden layer**).

Opisany program nie daje możliwości stosowania sieci więcej niż trójwarstwowej. Jest to wyraźne ograniczenie, które możesz jednak zmienić, jeśli tylko zechcesz „podłubać" w samym programie (masz przecież dostęp



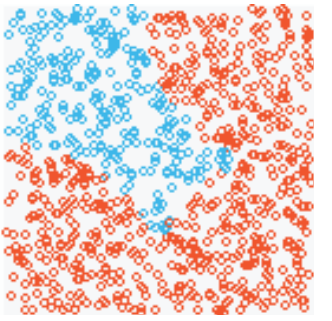

Rys. 8.8. Ustalanie struktury sieci w programie **Example 09** w przypadku wyboru tylko dwóch warstw

do kodu źródłowego)**.** Ograniczenie to podyktowane jest bowiem istotnymi względami natury obliczeniowej – program liczy się – jeśli wybierzesz dużą sieć – **wolno**. Przekonasz się, że wykonanie kilkuset kroków procesu uczenia (a podczas pracy z programem łatwo stwierdzisz, że kilkaset kroków to podczas obserwowania skutków uczenia wcale nie za dużo!) będzie trwało zauważalny moment – nawet jeśli masz nowy i szybki komputer.

## 8.4. Jak możesz tworzyć zadania rozpoznawania dla sieci?

Po wybraniu rozmiaru sieci i – jeśli trzeba – liczby neuronów w poszczególnych warstwach, następuje najciekawszy etap pracy z programem – formułowanie zadania, które sieć ma rozwiązywać.

Jak wiesz, istota zadania rozpoznawania polega na tym, że dla niektórych kombinacji sygnałów wejściowych sieć powinna reagować pozytywnie („rozpoznawać" odpowiednie sytuacje jako sympatyczne dla niej obrazy), zaś dla innych kombinacji reakcja sieci ma być negatywna (odrzucenie pewnych obrazów i odmowa ich rozpoznania jako tych, które sieć „lubi"). Informację, co sieć ma lubić, a co nie – zawiera ciąg uczący, złożony z pewnej liczby punktów, dla których ustalone są obie współrzędne. W ten sposób wiadomo, w którym miejscu taki punkt zaznaczyć na ekranie oraz wiadomo, jaka w tym punkcie jest oczekiwana poprawna odpowiedź sieci – pozytywna albo negatywna, co odpowiada czerwonym lub niebieskim punktom na ekranie (rys. 8.9).



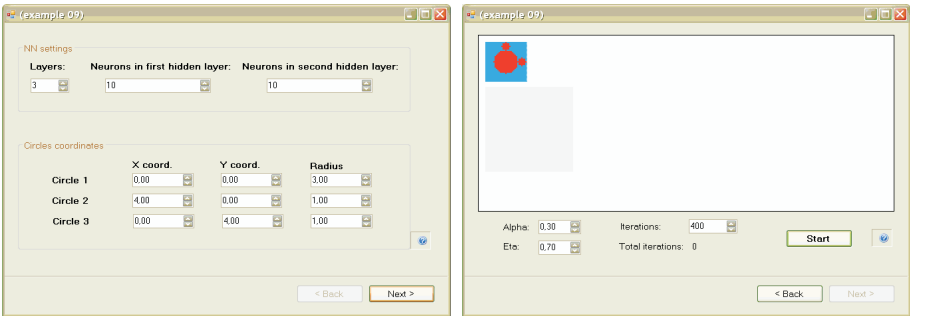
Rys. 8.9. Przykładowy ciąg uczący wygenerowany przez program **Example 09**

O tym, w których miejscach znajdą się punkty, dla których oczekiwana (pożądana) decyzja sieci ma być pozytywna, a gdzie są obszary, dla których odpowiedź sieci powinna być negatywna – **decydujesz Ty**. Mógłbym Ci pozwolić wprowadzać odpowiednie dane „z palca", na zasadzie trójek liczb – dwie wejściowe współrzędne i pożądana odpowiedź sieci – ale dla stworzenia sensownego zadania musiałbyś okropnie się napracować, a jego obraz, w postaci „mapy preferencji", mógłby być trudny do interpretacji.

Dlatego zdecydowałem, że ciąg uczący program **Example 09** będzie generował automatycznie, a Twoim zadaniem będzie wskazać mu, gdzie chcesz mieć pozytywne, a gdzie negatywne reakcje sieci. I znowu coś musiałem zadecydować **za Ciebie.** Mógłbym umożliwić Ci stworzenie całkiem dowolnych obszarów pozytywnych i negatywnych decyzji, które następnie podlegałyby procesowi uczenia i byłyby celem rozpoznawania dla sieci – ale takie założenie wymagałoby stworzenia w programie pełnego edytora graficznego, skomplikowanego w działaniu i trudnego w obsłudze. Dlatego wybrałem rozwiązanie kompromisowe – obszary pozytywnych decyzji, to znaczy fragmenty przestrzeni sygnałów wejściowych, w których sieć powinna odpowiadać „tak" (czerwone punkty na wykresie), stanowić będą **wnętrze trzech okręgów.** To dość duże ograniczenie, ale to pozwoli Ci szybko i wygodnie tworzyć zadania dla sieci i – dzięki temu, że będziesz mógł konstruować swoje obszary z **dowolnie wybranych** kół, o dowolnej wielkości i dowolnie rozmieszczonych – będziesz miał możliwość tworzenia naprawdę interesująco skomplikowanych zadań.

Współrzędne dla tych trzech okręgów możesz określić w tym samym oknie (pojawiającym się zaraz po uruchomieniu **Example 09**), w którym ustalałeś strukturę wybranej sieci (rys. 8.10a). Pierwsze dwie współrzędne x i y opisują miejsce, w którym ulokowany zostanie środek koła, a trzecia liczba określa jego promień (Radius). Ustalić je możesz dla każdego z trzech kół (**Circle 1, 2, 3**).

Wybierając te wartości masz całkowitą swobodę, ale – jak Ci już wyżej pisałem – w procesie rozpoznawania bierze udział tylko ta część płaszczyzny, dla której obie współrzędne mieszczą się w przedziale (–5,5). Oznacza to, że dość sensownym obszarem, w którym rozmieszczone są punkty wejściowe




a) b)

Rys. 8.10. a) Przykład ustalania struktury sieci oraz formowania obszarów dla rozpoznawania, b) „Mapka" obszarów, w których sieć (po nauczeniu) powinna dostarczać decyzji **pozytywnych** albo **negatywnych**

będące zbiorami cech obiektów przeznaczonych do pozytywnego rozpoznawania, jest – na przykład – okrąg o parametrach

**0,0,3**

natomiast okręgu o parametrach

**10,10,3**

w ogóle nie będziesz widział na ekranie i jego stosowanie jest pozbawione sensu.

Rezultat wyboru parametrów określających strukturę zadania, które ma być rozwiązywane przez sieć, będziesz mógł obejrzeć przechodząc do kolejnego panelu, w którym będzie prowadzona symulacja zachowania się Twojej sieci. Przechodzisz do niego klikając przycisk **Next** na dole okienka, patrz rys. 8.10a. Wyniki swoich decyzji będziesz mógł oglądać w postaci „mapki" obszarów, w których Twoja sieć (po nauczeniu) powinna dostarczać decyzji **pozytywnych** albo **negatywnych** (rys. 8.10b).

W ten sposób będziesz mógł w każdej chwili łatwo sprawdzić, czy Twoje przypuszczenia dotyczące kształtów zadawanych obszarów odpowiadają rzeczywistości – czy nie. Jeśli Ci się nie uda od razu wytworzyć zadania, które będzie odpowiadało Twoim potrzebom – możesz zawsze wrócić do poprzedniego panelu klikając przycisk **Back** (rys. 8.10b) i ponownie możesz podać w odpowiednich miejscach inne współrzędne kół i inne ich rozmiary.

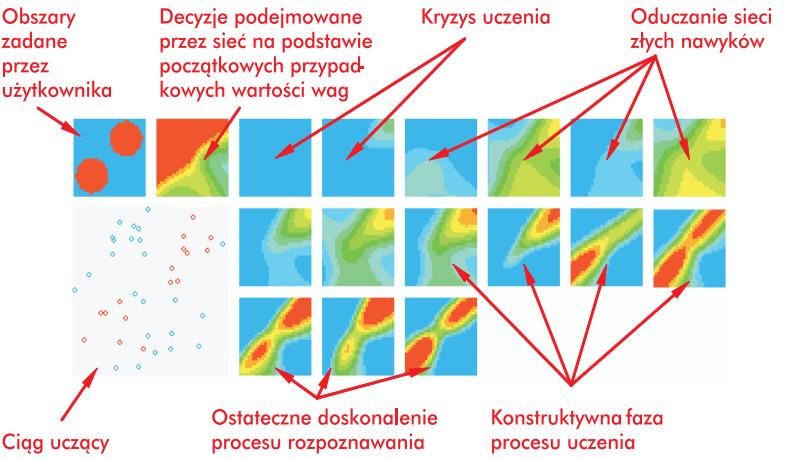
Uczenie sieci będzie się odbywało w ten sposób, że będą losowo wybierane z widocznego na ekranie obszaru kolejne punkty, następnie na podstawie Twojej mapki ustalane będzie, do której klasy punkty te należy zaliczyć, a potem zestaw złożony ze współrzędnych wylosowanego punktu i ustalonej poprawnej odpowiedzi sieci – będzie podawany do sieci jako element ciągu uczącego.

Zwróć uwagę na jeden łatwy do przeoczenia szczegół. Jeśli wybierzesz zadanie, w którym znacznie większy obszar w rozważanym okienku zajmować będą punkty, dla których wymagana będzie decyzja pozytywna (czerwone) – sieć w trakcie uczenia będzie stosunkowo rzadko otrzymywać przykłady obiektów, dla których powinna się nauczyć negatywnych reakcji i dlatego będzie się stosunkowo wolno uczyć. Identyczne kłopoty (tylko „w drugą stronę") pojawią się w przypadku sieci uczonej przykładami z **nadreprezentacją** przykładów, dla których wymagana jest reakcja negatywna („wielki błękit"). Dlatego należy dążyć do tego, by na wytworzonym rysunku było (w przybliżeniu) tyle samo czerwieni, co błękitu. Wtedy proces uczenia przebiegać będzie maksymalnie sprawnie – ku Twojej rzetelnej satysfakcji. Jeśli nie posłuchasz mojej rady – otrzymasz zadanie, którego sieć też się w końcu nauczy, ale co się przy tym naczekasz – to Twoje!

Jak już wytworzysz zadanie w postaci mapy obszarów satysfakcjonującej Twoje ambicje i dobrze zrównoważonej (w wyżej omówionym sensie), mo-

żesz rozpocząć proces uczenia. W tym celu kliknij przycisk **Start** widoczny na rysunku 8.10 b.

Na początku program pokaże Ci, jakie decyzje podejmuje sieć jeszcze przed rozpoczęciem procesu uczenia, na podstawie przypadkowych (losowo wybranych) wartości współczynników wag wszystkich wchodzących w jej skład neuronów. Na ogół ten początkowy rozkład decyzji produkowanych przez sieć ma się nijak do zadania, które sieci postawiłeś – no bo niby skąd ta sieć miałaby **przed uczeniem** wiedzieć, czego Ty od niej chcesz? Warto się jednak przyjrzeć tej początkowej mapie „wrodzonych preferencji" sieci i porównać ją z obrazkiem, jaki obok sam wytworzyłeś definiując zadanie do rozwiązania. Cały dalszy przebieg procesu uczenia będzie bardzo silnie zależny od tego, jak bardzo te dwa obrazki są do siebie podobne. Jeśli oba obrazki **są** podobne, to sieć będzie mogła uczyć się szybko i wydajnie. Jeśli natomiast występują duże różnice (na ogół tak właśnie bywa), to sieć uczy się wolno i (szczególnie w początkowym okresie nauki) nie widać szybkich postępów, gdyż trzeba najpierw pozbyć się szkodliwych „wrodzonych nawyków" (oj, ciężko to sieci przychodzi, ciężko!), a dopiero potem nastąpi właściwy proces nabywania i doskonalenia nowych umiejętności (rys. 8.11).




Rys. 8.11. Sposób, w jaki program **Example 09** prezentuje kolejne etapy procesu uczenia

Powiedzmy teraz kilka słów na temat struktury produkowanego przez program i wyświetlanego na ekranie rysunku. Widoczne na nim (tworzone w trakcie symulacji) wypełnione barwnymi plamami małe kwadraty stanowią obraz „stanu świadomości" sieci neuronowej w kolejnych etapach procesu uczenia. **Każdy punkt wewnątrz każdego kwadratu** symbolizuje – jak już wiesz – zespół dwóch danych (odpowiadających poziomej i pionowej współ-

rzędnej położenia punktu), stanowiących sygnały wejściowe dla sieci neuronowej, a kolor odpowiedniego punktu pokazuje, jaka w określonych warunkach była odpowiedź symulowanej sieci. Trzeba zacząć od rozważenia znaczenia pierwszych dwóch kwadratów, leżących w lewym górnym rogu rysunku. Pierwszy z nich pokazuje, czego usiłujesz swoją sieć nauczyć. Postępując w sposób opisany w poprzednim podrozdziale, z góry zakładasz, jakich warunków modelowane „zwierzę" ma poszukiwać, a od jakich uciekać – co na rysunku wyrażone jest w postaci mapy pożądanych zachowań. Drugi wyróżniony kwadrat (drugi od lewej w górnym rzędzie) pokazuje naturalne („wrodzone") skłonności sieci. W każdym kolejnym eksperymencie sieć może mieć inne początkowe (przypadkowo nadawane) wartości swoich parametrów, w związku z tym jej początkowe zachowanie jest niemożliwe do przewidzenia.

Kolejne obrazki, odpowiadające kolejnym fazom procesu uczenia, ilustrowanym na ekranie w postaci kolejnych kwadratów widocznych na rysunku 8.11, następować mogą po sobie w różnych odstępach, gdyż każdorazowo Ty sam będziesz mógł decydować, ile kroków uczenia ma wykonać sieć, zanim po raz kolejny pokaże wyniki (służy do tego pole **Iterations**). Zapewne zauważyłeś, że pod polem **Iterations** znajduje się dodatkowe pole **Total Iterations** – jest to pole informacyjne i pokazywać ono będzie, ile w sumie kroków uczenia wykonano we wszystkich fazach do bieżącego momentu.

Przed wykonaniem podanej przez Ciebie (lub pozostawionej, niezmienionej z poprzedniej fazy) liczby kroków uczenia możesz również modyfikować wartości współczynników warunkujących szybkość uczenia (*współczynnik uczenia – **Alpha*** i *momentum – **Eta**),* bo odpowiednie okienka dla tych wartości są cały czas widoczne na ekranie. Możesz je zadawać całkiem dowolnie, ale najmądrzej zrobisz, jeśli na początku nie będziesz zmieniał tych parametrów w trakcie uczenia i pozostawisz takie ich wartości, jakie program sam proponuje. Potem, gdy już poznasz uroki (i przykrości...) typowego procesu uczenia – możesz podjąć próbę modyfikacji tych parametrów, badając, jakie będą tego skutki.

## 8.5. Jakie formy uczenia można zaobserwować w sieci?

Opisany program **Example 09** daje Ci okazję do bardzo ciekawych badań nad formami uczenia, jakie mogą pojawiać się w sieciach neuronowych. Badania te są bardzo ciekawe i inspirujące, więc sam spędziłem wiele godzin dobierając różne zadania i przeprowadzając proces uczenia z różnymi parametrami. Pozwól, że podzielę się z Tobą teraz pewnymi moimi spostrzeżeniami. Później to Ty mi będziesz mógł opowiadać, jak przebiegał proces uczenia rozważanych przez Ciebie sieci.

Uczenie żywych organizmów, a zwłaszcza mechanizmy neuronowe warunkujące to uczenie, były „od zawsze" przedmiotem zainteresowania przyrodników. Gromadząc i systematyzując przez wiele lat obserwacje dotyczące różnych form uczenia się ludzi i zwierząt zebrano obszerną wiedzę, która pozwoliła wykryć podstawowe prawidłowości tego procesu. Wiedza ta znalazła między innymi zastosowanie w dydaktyce ludzi i przy tresurze zwierząt. Jest to jednak wiedza behawioralna – o zachowaniach, a nie o mechanizmach przyczynowych. Kiedy bowiem następnie usiłowano odkryć **istotę** procesu uczenia, wykonując tysiące doświadczeń neurofizjologicznych i biochemicznych – sukces był znacznie skromniejszy. Na szczęście pojawiły się obecnie możliwości weryfikacji koncepcji dotyczących różnych form działania systemu nerwowego, także w trakcie uczenia się i gromadzenia wiedzy, właśnie za pomocą ich symulacji przy zastosowaniu sieci neuronowych. Używa się do tego takich właśnie programów, jak omawiany w tym rozdziale **Example 09**. Wykonuje on symulacje procesu uczenia i produkuje rysunki, których obejrzenie może dostarczyć wielu bardzo ciekawych refleksji. Zajmiemy się dokładniej jednym takim przypadkiem, pozostałe poznasz i przebadasz sam!

Jak pamiętasz, przebieg uczenia sieci neuronowej polega na prezentowaniu na wejściach sieci przypadkowo dobieranych kombinacji sygnałów wejściowych i na wymuszaniu na jej wyjściu takiego zachowania, jakie przewiduje mapa pożądanych zachowań. Scenariusz ten, sprowadzony do realiów rozważanego tu przykładu „zwierzęcia" posiadającego dwa narządy zmysłów, polega na wielokrotnym umieszczaniu naszego „zwierzęcia" w różnych „środowiskach", w wyniku czego do wejść sieci (receptorów „zwierzęcia") dostarczane są przypadkowe (ale znane) sygnały. Sieć reaguje tak, jak jej nakazuje aktualnie zawarta w niej wiedza – czyli jedne warunki aprobuje, inne nie. Natomiast „nauczyciel" (czyli komputer prowadzący trening), mając mapę pożądanych zachowań sieci, podaje jej sygnał wzorcowy, tak jak gdyby mówił: *to Ci się ma podobać, a tamto nie!*

Przebieg tego treningu możesz obserwować dynamicznie (na ekranie komputera), gdyż pełny pogląd na temat jego przebiegu daje obrazek w postaci dużego kwadratu w lewym dolnym rogu rysunku wyświetlanego przez program na Twoim ekranie. W kwadracie tym zapalane są kolejno punkty odpowiadające zestawom tych sygnałów wejściowych, które podawane są podczas uczenia do trenowanej sieci. Widać, że są one losowo rozsiane w całym obszarze. Widać też, że każdy punkt jest zaznaczony kolorem – czerwonym albo niebieskim. Jest to właśnie podpowiedź nauczyciela: to jest dobre, a tamto nie. Sieć uwzględnia te podpowiedzi i koryguje swoje błędy, zmieniając wartości swoich parametrów (wag synaptycznych), przy czym robi to w sposób pokazany wcześniej podczas prezentacji działania algorytmu wstecznej propagacji błędów.

Co pewien czas proces uczenia jest przerywany i sieć poddawana jest „egzaminowi" – musi podać (dla wszystkich możliwych punktów) swoje oceny. Wyniki tych egzaminów prezentowane są przez program w postaci „map" kolorowych punktów, stanowiących zawartości kolejnych kwadratów, wyświetlanych (w kolejności od lewej do prawej, jak w komiksie) najpierw w górnym, a potem w dolnym wierszu zbiorczego obrazka produkowanego przez program **Example 09**. Ty będziesz każdorazowo decydować, ile kroków uczenia trzeba wykonać, zanim poddasz sieć kolejnemu egzaminowi.

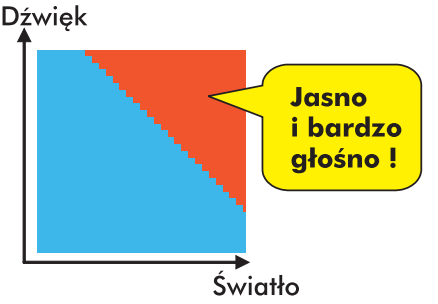
Przykład form uczenia, od którego zacznę prezentację, dotyczyć będzie jednowarstwowej sieci o strukturze

**2 – 1**.

Sieci tej postawiono bardzo proste zadanie, wyrażające się w postaci podzielenia obszaru wszystkich możliwych danych wejściowych na strefę aprobowaną i strefę nie aprobowaną za pomocą linii (prawie) prostej. Aby to uzyskać, należy podać podczas konstrukcji zadania testowego następujące współrzędne kół[1]:

**100,100,140**
**0,0,0**
**0,0,0**

Powstała wtedy mapa „żądanych" przez nauczyciela form zachowania sieci (rys. 8.12) ma prosty i łatwy do interpretacji kształt – zwierzę powinno pożądać i szukać środowiska, w którym jest jasno i głośno.




Rys. 8.12. Obraz modelu środowiska i zadawanych przez nauczyciela preferencji

Jak już wiesz, program **Example 09** działa w ten sposób, że na początku przeprowadzony zostanie test naturalnych („wrodzonych") preferencji sieci. Na ekranie pojawi się drugi kwadrat (por. rys. 8.13), który trzeba interpretować jako kompletny wynik zestawu testów, jakie program przeprowadza na symu-

[1] Na uwagę zasługuje tu „sztuczka" z podawaniem współrzędnych kół o zerowych promieniach stosowanych w przypadku, gdy jedno koło wystarcza i nie potrzebujesz już dalszej komplikacji struktury zadania.

lowanym „zwierzęciu" przed jego uczeniem. Po prostu program umieszcza je po kolei we wszystkich możliwych środowiskach (bada wszystkie punkty wewnątrz kwadratu) i w każdym z nich sprawdza reakcję sieci. Tam, gdzie reakcja jest pozytywna – odpowiedni punkt zabarwiany jest na kolor czerwony. Tam, gdzie reakcja jest negatywna – punkt przybiera barwę błękitną. Wreszcie stany pośrednie sygnalizowane są za pomocą pośredniej skali barw. Widać, że „zwierzę" przed rozpoczęciem procesu uczenia „lubi" mroczne zakamarki, najlepiej czuje się w ciemności i w ciszy, ale jest skłonne tolerować trochę światła, ogólnie tym więcej, im jest głośniej. Mówiąc żartem – jest to jak widać zwolennik dyskoteki i sypialni! (Ale **zwierzak** nam się trafił ☺ …)

Jeśli powtórzysz opisane tu eksperymenty na swoim komputerze, to niewątpliwie uzyskasz inny obraz stanu początkowych preferencji sieci, ponieważ – jak już zapewne zdążyłeś się przyzwyczaić – jest on **przypadkowy.** Szczegóły nie są jednak ważne. Widać, że stan pożądany (z punktu widzenia nauczyciela) i stan rzeczywisty znacznie różnią się od siebie (tak jest w opisywanych tu badaniach prawie zawsze), i dlatego podejmowane jest intensywne uczenie sieci.

Rys. 8.13. Prezentacja pożądanych przez nauczyciela i „wrodzonych" właściwości sieci

Kolejne kroki procesu uczenia inicjujesz, podając, ile kroków uczenia chcesz wykonać, zanim program znowu pokaże Ci mapę rozkładu decyzji podejmowanych przez sieć. Radzę Ci, żebyś w małych sieciach stosował „podglądanie" efektów uczenia stosunkowo często, na przykład co 10 kroków. Natomiast dla sieci dużych, o wielu elementach w warstwach ukrytych – trzeba większej porcji nauki, żeby wyniki stały się zauważalne. Wystarczy więc tylko podawać długości kolejnych epok procesu uczenia, a potem siedzieć, patrzeć, porównywać i wyciągać wnioski. W ten sposób można dowiedzieć się znacznie więcej na temat sieci niż przez studiowanie książek czy artykułów naukowych!

Proponuję, żebyś teraz uważnie przejrzał kolejne etapy uczenia z kilku przykładów, które ja przerobiłem przygotowując tę książkę – chociaż zapewne Ty usiłując iść moim tropem uzyskasz na swoim komputerze nieco inne obrazy. Przynajmniej będziesz wiedział, czego się spodziewać!

Po pierwszym etapie uczenia wyświetlony kwadrat (patrz rysunek 8.14) pokazuje obraz bardzo podobny do tego, który odpowiadał początkowym („wrodzonym") właściwościom systemu. Oznacza to, że mimo intensywne-

go treningu sieć nie chce „wyrzec się swoich przekonań". Takie początkowe trwanie w błędach jest dosyć typowe dla uczenia sieci neuronowych i znajduje także swoje potwierdzenia w uczeniu ludzi i zwierząt.

Rys. 8.14. Stan po pierwszym okresie uczenia sieci

Następny etapu uczenia wnosi do rysunku kolejny kwadrat i pokazuje pewien pośredni etap procesu uczenia. Sieć zaczyna w zauważalny sposób zmieniać swoje zachowanie, ale jest jeszcze daleka od ostatecznego rozwiązania – linia graniczna między obszarem pozytywnych i negatywnych decyzji przebiega prawie poziomo (rys. 8.15). Sieci w tym momencie **wydaje się**, że wie, o co chodzi nauczycielowi: ma po prostu lubić, gdy jest głośno. Niestety, to jest oczywiście błąd, więc konieczne będą dalsze korekty.

Rys. 8.15. Kolejny etap procesu uczenia

Dopiero po kolejnej porcji nauki charakter zachowania sieci zaczyna być podobny do pożądanego, chociaż dokładne położenie linii granicznej odbiega jeszcze nieznacznie od idealnego (rys. 8.16).

Rys. 8.16. Po trzech etapach uczenia sieć jest bliska sukcesu

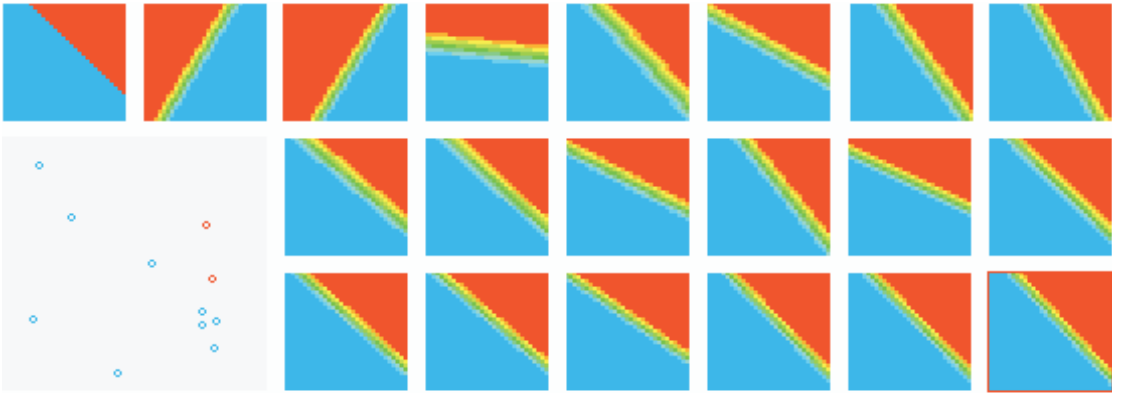
Wynik można by uznać za znakomity – szczególnie biorąc pod uwagę fakt, że został on osiągnięty w bardzo krótkim czasie. W zasadzie na tym uczenie należało przerwać, ale dla potrzeb eksperymentu założono, że surowy nauczyciel wymaga absolutnej doskonałości, wymuszając poprawki w działaniu sieci po każdym, nawet drobnym błędzie.

Efektem jest gwałtowny kryzys. Sieć **pogarsza** swoje zachowanie (położenie linii granicznej po czwartym etapie procesu uczenia jest gorsze niż osiągnięte wcześniej – por. rys. 8.17).

Rys. 8.17. Kryzys w procesie uczenia, będący skutkiem nadmiernej surowości nauczyciela

Trzeba potem dość długiego procesu, obejmującego wiele etapów uczenia, by sieć osiągnęła wymaganą doskonałą zgodność swego zachowania z podawanymi przez nauczyciela wzorcami. Warto odnotować i zapamiętać ten efekt: Nadmierna surowość nauczyciela na etapie, kiedy wiedza sieci nie jest jeszcze całkiem pewna i ustabilizowana, prawie zawsze przynosi szkody, czasem nawet w postaci całkowitego „załamania się" ucznia.

Całkowity obraz wyników procesu uczenia tej prostej sieci tego prostego zadania, widoczny na ekranie po wykonaniu 12 etapów uczenia, przedstawia rysunek 8.18.

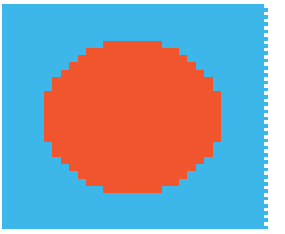


Rys. 8.18. Końcowy wynik procesu uczenia sieci

Przerwijmy działanie programu i powróćmy (przycisk **Back**) do ekranu, w którym ustalamy nowe parametry zadania. Po podaniu takiej samej jak poprzednio (jednowarstwowej) struktury sieci neuronowej sformułujmy dla sieci inne (trudniejsze!) zadanie. Po prostu zażądajmy, żeby modelowane zwierzę stało się zwolennikiem „złotego środka" – wszelkie skrajne sytuacje mają być odrzucane i modelowane zwierzę ma pragnąć i pożądać warunków przeciętnych. Łatwo to uzyskać, wpisując w odpowiednie okienka współrzędne jednego koła i jego środek:

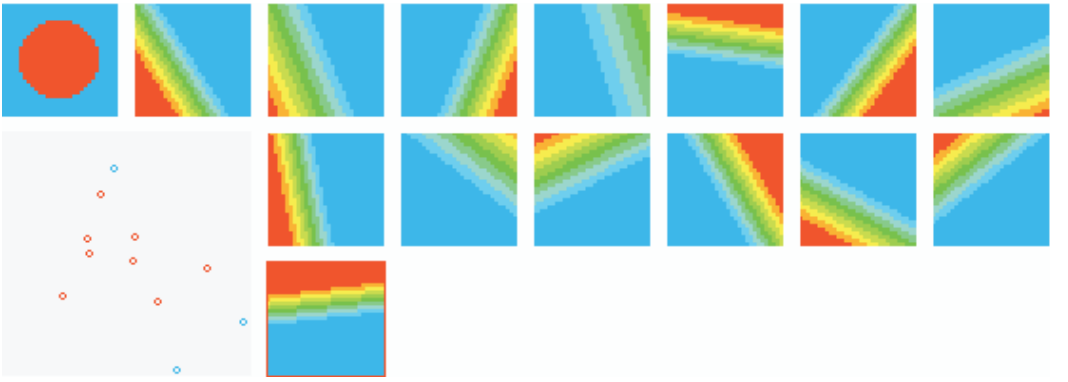
**0,0,3.5**

Pozostałe dwa koła znowu zerujemy, podając parametry:
**0,0,0**
**0,0,0**
Obraz wzorcowego zachowania sieci, jaki tym razem uzyskamy, pokazuje rysunek 8.19.



Rys. 8.19. Wzorzec nowego zadania dla sieci

Obserwując zachowanie sieci na rysunku 8.20 (zbudowanym analogicznie jak rysunek 8.18), widzimy, jak miota się ona i zmienia swoje zachowanie usiłując uniknąć kar stosowanych przez nauczyciela – nie ma jednak szans.



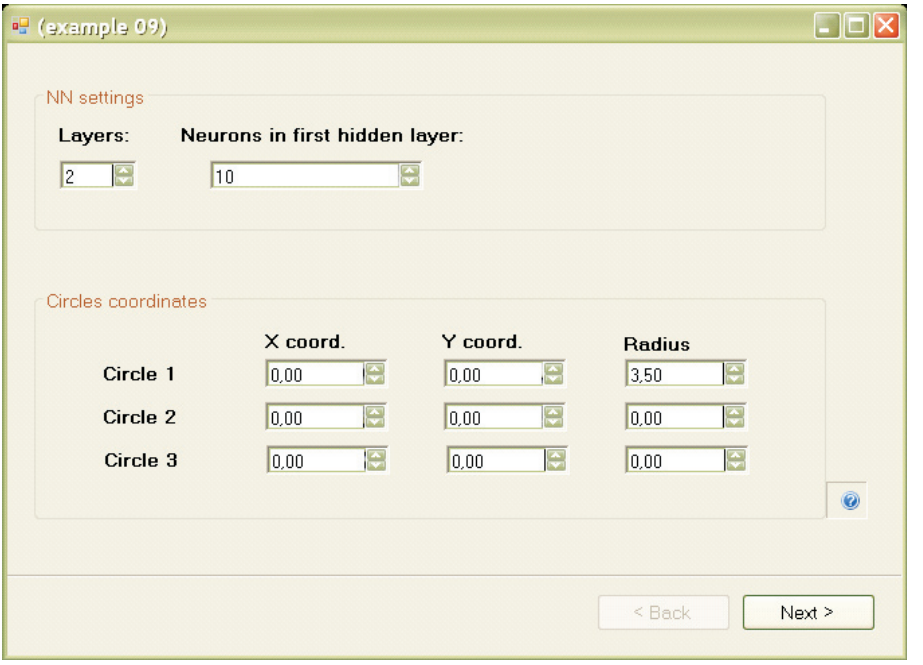
Rys. 8.20. Niepowodzenie uczenia w przypadku zbyt ubogiej struktury sieci

Jakkolwiek bowiem dobrane zostaną parametry sieci i jakkolwiek będzie przebiegała granica między obszarem pozytywnych i negatywnych reakcji – sieć zawsze wpadnie w pułapkę związaną z tym, że będzie traktowała jako przyjazne środowisko o jakichś krańcowych właściwościach. Na przykład na wielu kolejnych rysunkach widać, jak modelowane „zwierzę" usiłuje ukryć się w ciemności, wybierając jako główną wskazówkę niewielką ilość światła w kilku podanych przez nauczyciela przykładach – i za to właśnie będzie w trakcie dalszej nauki karane. Zanim przerwałem eksperyment – zwierzę „sformułowało" kolejną niepoprawną hipotezę – zaczęło unikać obszarów, w których panowała złowroga cisza, sądząc, że istotą zadania, którego nie

umie rozwiązać jest to, że powinno poszukiwać zgiełku. Niestety, to także był błąd…

Powód niepowodzenia jest prosty – modelowana sieć neuronowa była zbyt prymitywna, żeby nauczyć się tak złożonej formy zachowania, jak „wybór złotego środka". Dla używanej w doświadczeniu sieci jedyną znaną formą różnicowania wejściowych sygnałów była ich liniowa dyskryminacja – a to okazało się w rozważanym tu przykładzie zbyt prymitywne i zbyt ubogie.

Dlatego kolejny eksperyment polegać będzie na zastosowaniu do tego zadania sieci o większej liczbie elementów i bardziej skomplikowanej strukturze. Ustalając nową strukturę sieci, można w tym przypadku podać dane takie, jak na rysunku 8.21.




Rys. 8.21. Sposób ustalenia wartości dla „bardziej inteligentnej" sieci

Warto zauważyć, że po zadeklarowaniu, że sieć ma więcej niż jedną warstwę – program zadał dodatkowe pytanie o liczbę neuronów w tej dodatkowej warstwie! Gdy będzie trzeba zbudować jeszcze większe sieci – dodatkowych pytań będzie więcej.

Jak wiesz, im więcej warstw i im więcej neuronów ma sieć, tym większy będzie jej „potencjał intelektualny". Taka sieć naprawdę potrafi lepiej wykorzystać wbudowaną większą „wrodzoną inteligencję". Popatrzmy teraz, jak

taka sieć się uczy. Początkowy rozkład barwnych plam pokazanych na rysunku 8.22 wskazuje, że sieć ta ma początkowo – sama z siebie – stosunek entuzjastyczny do wszystkiego. Modelowane „zwierzę" czuje się świetnie w prawie wszystkich środowiskach, najmniej jednak kocha ciemne i ciche zakamarki.

Rys. 8.22. Przykładowy stan początkowy bardziej złożonej sieci

Pierwsze doświadczenia zdobyte podczas procesu uczenia pokazują, że świat nie jest wcale taki wspaniały i poza słodkimi migdałami zawiera także pokrzywy. Sieć reaguje w sposób typowy – pogłębia z początku swoje pierwotne uprzedzenia i po pierwszym etapie uczenia zdecydowanie mniej lubi ciemne zakamarki – ale jeszcze jest bardzo daleka od końcowego sukcesu (rys. 8.23).

Rys. 8.23. Początkowy etap uczenia sieci

Dalsze uczenie prowadzi jednak naszego „zwierzaka" do kolejnych rozczarowań. Mnożą się kary za nadmierną ufność i przesadny entuzjazm, na co sieć reaguje wycofując się w kierunku coraz bardziej nieufnego stosunku do podstępnego świata. Widać, jak po drugim etapie uczenia z początkowego entuzjazmu i sympatii do wszystkich i wszystkiego pozostaje tylko lekkie zamiłowanie do cichych słonecznych zakątków (mała plamka żółtego koloru u dołu i po prawej stronie ostatniego kwadratu na rysunku 8.24).

Rys. 8.24. Kolejny etap uczenia

Ale to także spotyka się z krytyką nauczyciela, więc po kolejnym etapie uczenia sieć popada w totalną negację – absolutnie **nic** jej się nie podoba! Taki stan absolutnego załamania i zniechęcenia jest bardzo typowy dla większości znanych eksperymentów z uczeniem sieci neuronowej, i poprzedza zwykle próbę konstrukcji pozytywnego obrazu wymaganej przez nauczyciela wiedzy (rys.8. 25).

Rys. 8.25. Kryzys totalnej negacji w procesie uczenia sieci

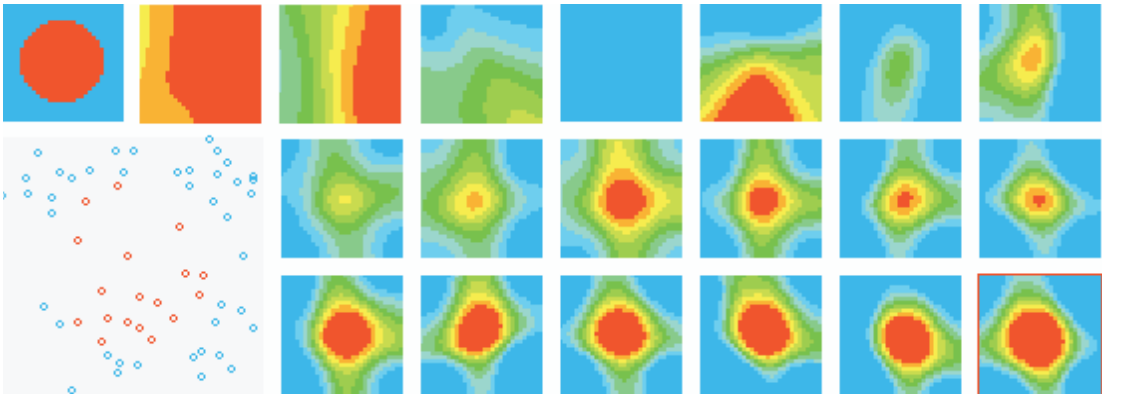
Taką próbę obserwujemy na rysunku 8.26 pokazującym czwarty krok procesu uczenia – wystarcza kilka pokazanych przez nauczyciela przykładów sytuacji, na które sieć powinna reagować pozytywnie – a pojawia się kolejna faza optymizmu i entuzjazmu, wyrażająca się aprobatą dla prawie wszystkich środowisk z wyjątkiem takich, które są bardzo głośne, w szczególności gdy są dość przeciętnie oświetlone. Całkowita ciemność albo jaskrawe światło zmniejszają tolerancję na hałas.

Rys. 8.26. Pozytywna fala optymizmu

Oczywiście, kolejny etap procesu uczenia musi te nadmiernie wybujałe nadzieje poskromić, w wyniku czego sieć ponownie wycofuje się w obszar negacji i frustracji – ale pozostaje pewien drobny ślad tych pozytywnych doświadczeń z przeszłości, które nie były przyczyną dotkliwych kar (zielony „jęzor" w okienku ilustrującym stan sieci po pięciu cyklach uczenia), który już w następnym kroku przekształca się w zalążek prawidłowej hipotezy (czerwona plama w okolicy centrum przedziału zmienności – rys. 8.27).

Rys. 8.27. Powstanie poprawnej hipotezy roboczej

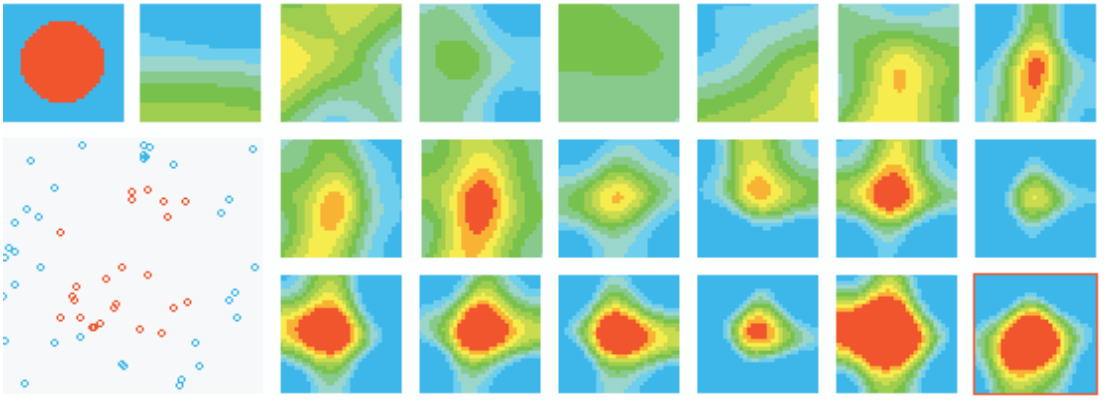
Dalszy proces uczenia służy już tylko do tego, żeby powściągnąć kolejne przypływy entuzjazmu i sprowadzić obszar pozytywnych reakcji sieci do właściwego rozmiaru. W momencie przerywania procesu uczenia (po 15 etapach) sieć umie już stosunkowo dobrze odtwarzać podawaną przez nauczyciela umiejętność klasyfikacji sygnałów i dalsze doskonalenie tej umiejętności nie jest już konieczne (rys. 8.28).



Rys. 8.28. Pełny przebieg procesu uczenia sieci bardziej złożonego zadania

Jak widać – system mający „mózg" większy (ponad dziesięciokrotnie więcej neuronów!) i sprawniejszy (ciekawsza struktura połączeń) okazał się zdolny do nauczenia takiej formy zachowania, której prymitywniejszy twór posiąść nie był w stanie.

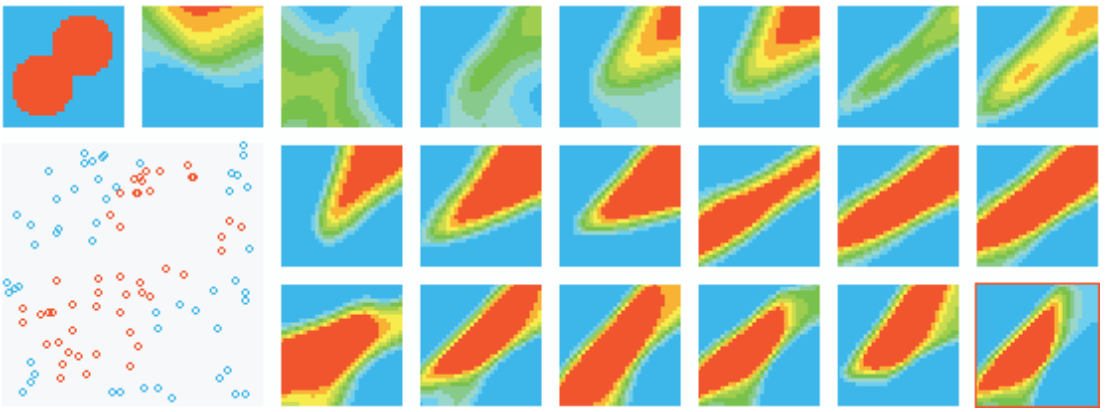
Kolejne ciekawe doświadczenie ilustruje rysunek 8.29. Na rysunku tym pokazano, jak dokładnie tej samej umiejętności (tzn. wykrywania, że nauczyciel życzy sobie, by pozytywnie reagować na przeciętne warunki oświetlenia i głośności, odrzucając wszelkie skrajności) – uczy się inny egzemplarz sieci neuronowej o tej samej strukturze, jak sieć, której uczenie pokazano na rysunku 8.28. Jedyna różnica między sieciami, których formy uczenia pokazano na rysunkach 8.28 i 8.29, polega na tym, że sieci te mają inne (przypadkowe) wartości początkowe parametrów wewnętrznych, co powoduje diametralnie inne „wrodzone preferencje" i zmienia zasadniczo przebieg procesu uczenia.



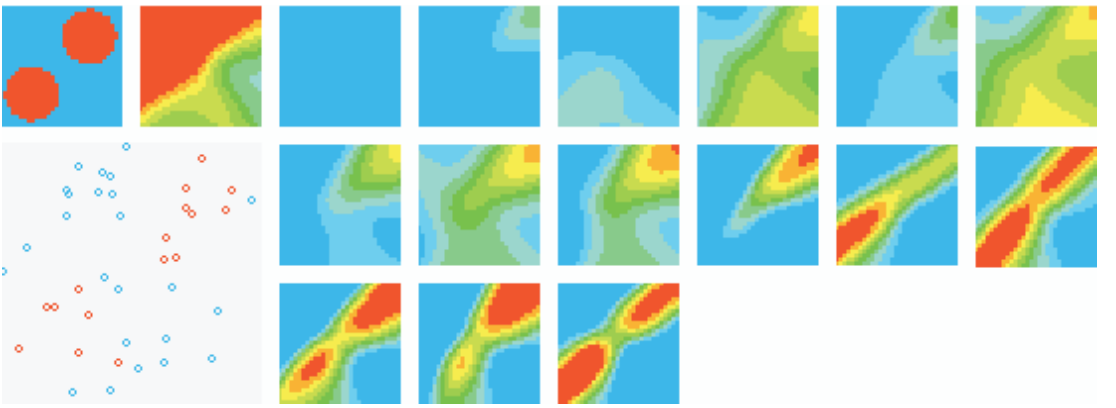
Rys. 8.29. Przebieg uczenia innego egzemplarza sieci

Ta druga sieć, której uczenie pokazano na rysunku 8.29, mimo bliźniaczego podobieństwa struktury do sieci związanej z rysunkiem 8.28, ma z początku skłonności raczej melancholiczne. Nic się jej tak naprawdę nie podoba – na rysunku dominują chłodne zielenie i błękity. Początkowy okres uczenia powoduje nikły wzrost zainteresowania „warunkami dyskotekowymi" (plama żółtej barwy w kwadracie odpowiadającym pierwszemu etapowi procesu uczenia), ale kilka kolejnych niepowodzeń w procesie uczenia wtrąca sieć ponownie w obszar totalnej (chociaż umiarkowanej) negacji. W czwartym etapie uczenia sieć usiłuje postawić hipotezę, że nauczycielowi zależy na aprobowaniu jasnych pomieszczeń o umiarkowanym poziomie dźwięku, jednak już na kolejnym etapie procesu uczenia pojawia się przypuszczenie, że chodzi o afirmację warunków, które można uznać za średnie i przeciętne. Przypuszczenie to z etapu na etap ulega wzmocnieniu i konkretyzacji. Wyraża się to faktem, że na rysunku pojawia się czerwona barwa kategorycznej aprobaty, której obszar z kroku na krok staje się coraz lepiej zgrany z rejonem, który w przewidzianym przez nauczyciela wzorcu zachowania odpowiada obszarowi pożądanej afirmacji, zaś wyrażające niepewność i wahanie barwy zielona i żółta zanikają na rzecz coraz dokładniej krystalizującego się podziału na warunki zdecydowanie dobre i zdecydowanie złe. W końcu sieć „melancholiczna" uczy się działać praktycznie tak samo, jak sieć „entuzjastyczna", co dowodzi, że „wrodzone predyspozycje" nie są najważniejsze – wytrwałą nauką można zawsze osiągnąć cel, chociaż droga dochodzenia do wiedzy może być odmienna dla różnych punktów startowych.

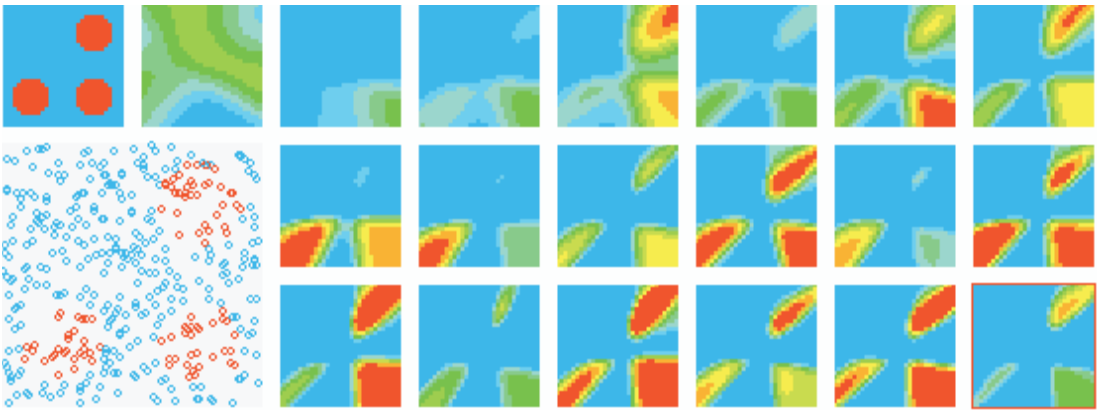
Posługując się zaproponowanym programem, można wykonać bardzo wiele doświadczeń. Na rysunkach 8.30–8.32 pokazano przebieg niektórych z nich. Zastanowienie się nad przebiegiem pokazanych na tych rysunkach



Rys. 8.30. Przykład zadania dla dwuwarstwowej sieci



Rys. 8.31. Przykład zadania, które nie każda dwuwarstwowa sieć zdoła rozwiązać



Rys. 8.32. Zadanie o dużym stopniu trudności i wyłącznie dla trójwarstwowej sieci

procesów uczenia, odtworzenie ich na swoim komputerze, wykonanie innych własnych badań – to fascynujący teren dla wielu odkryć. Warto na ten teren odważnie wkroczyć!

I jeszcze jedno spostrzeżenie na koniec tego rozdziału. Porównaj proszę rysunki 8.28–8.32 z rysunkiem 8.20. Możesz zauważyć, jak odmienne jest zachowanie się sieci o dużych „możliwościach intelektualnych" i prymitywnej sieci, która absolutnie nie była zdolna do nauczenia się bardziej złożonych zasad działania. Zauważ, że ten neuronowy „głupol" nie był w stanie nauczyć się poprawnej reakcji, jednak mimo to jego reakcje były w każdym momencie bardzo kategoryczne i zdecydowane. Dla niego cały czas świat był bardzo „czarno-biały" – „to" absolutnie dobre, a „tamto" zdecydowanie złe. Jeśli doświadczenie podczas uczenia nie potwierdzało tych kategorycznych sądów – były one zastępowane innym sądem, równie kategorycznym, chociaż często

równie nietrafnym. Natomiast sieci mające większe „możliwości intelektualne" wykazywały zdolność do subtelniejszych rozróżnień, dochodziły do rozwiązania wolniej i przechodziły przez wiele etapów naznaczonych rozterkami, załamaniami, i „hamletyzowaniem". Jednak to właśnie te niezdecydowane, neurotyczne, nadmiernie inteligentne osobniki zyskiwały sukces w postaci ostatecznego prawidłowego dopasowania swojego zachowania do „reguł gry" narzucanych przez nauczyciela, podczas gdy kretynowaty „twardziel" kręcił się w kółko nie mogąc dojść do żadnych sensownych konkluzji...

Zauważ, jak ciekawe i różnorodne zachowania sieci neuronowych można było obserwować z pomocą tego jednego prościutkiego programu. Komputer modelujący sieć neuronową nie zachowywał się jak tępa i ograniczona maszyna, nie działał jednostajnie i w sposób powtarzalny – przeciwnie, wykazywał pewne cechy indywidualne, zmienność nastrojów, zróżnicowane uzdolnienia... Czy czegoś Ci to nie przypomina?

## 8.6. Co jeszcze można zaobserwować w badanej sieci?

Mógłbym w zasadzie na tym poprzestać, pozwalając Ci odkryć wszystkie ciekawostki, jakie potencjalnie tkwią w tym programie, jednak spróbuję podpowiedzieć Ci, jakie obszary warto zastosować na początku, zanim nie nabierzesz wprawy. Z moich doświadczeń wynika, że ciekawy przebieg procesu uczenia i rozpoznawania możesz uzyskać stosując „bałwanka" (patrz rys. 8.10) złożonego z okręgów o parametrach

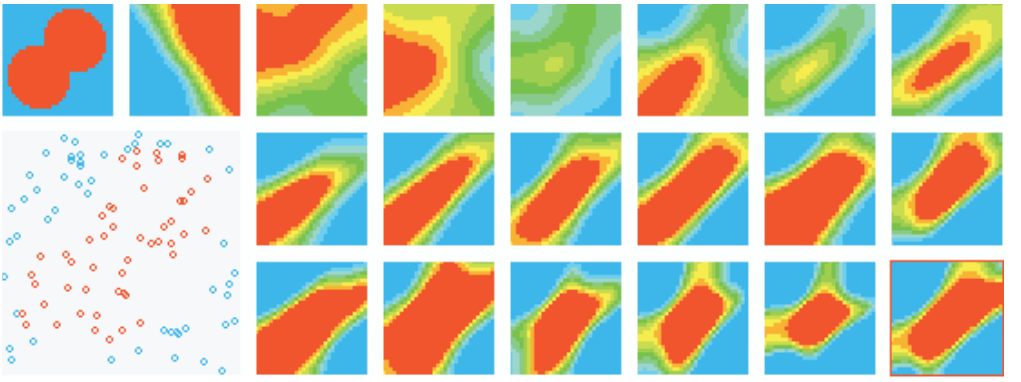
**0,0,3**
**4,0,1**
**0,4,1**

Jest to zadanie już wystarczająco trudne, żeby sieć musiała się srodze napracować, zanim odkryje prawidłową zasadę rozpoznawania (sieć dla tego zadania powinna być trójwarstwowa, a liczba elementów w warstwach ukrytych powinna być spora – na przykład 7 elementów w pierwszej warstwie ukrytej i 5 w drugiej). Równocześnie zadanie to jest wystarczająco zwarte, by dobrze się prezentowało na ekranie i by łatwo można było oceniać produkowane przez sieć wyniki. Nie pokażę Ci wyników, jakie uzyskiwałem dla tych przykładów, bo nie chcę Ci psuć przyjemności ich samodzielnego odkrywania. Pokażę Ci jednak, jakie wyniki możesz uzyskać, gdy zdecydujesz się wykonać z pomocą opisanego programu trochę bardziej wnikliwe badania właściwości sieci i jej podatności na niektóre parametry, determinujące przebieg procesu uczenia.

Jak zapewne już doskonale wiesz, przed wykonaniem podanej przez Ciebie liczby kroków uczenia program umożliwia Ci zmianę wartości współczynników warunkujących szybkość uczenia. Pierwszy z nich, oznaczony na panelu

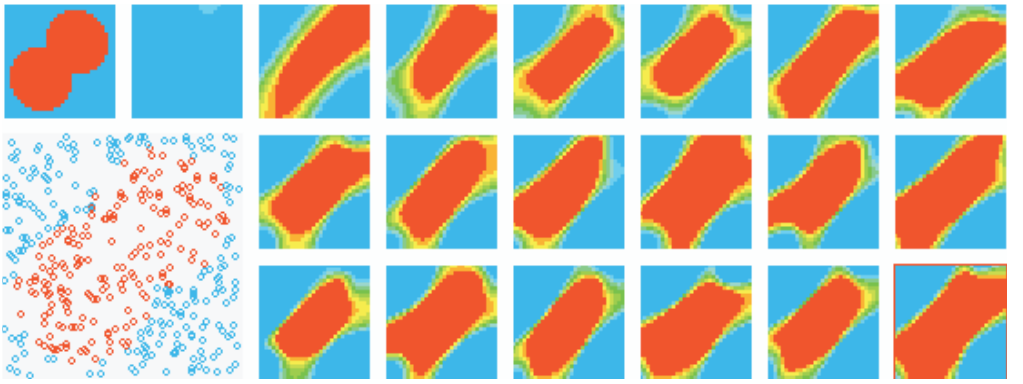
sterującym pracą programu jako *Alpha,* jest to *współczynnik uczenia* nazywany w literaturze międzynarodowej jako *learning rate.* Ustawiając większe wartości tego współczynnika powodujesz, że uczenie sieci będzie bardziej intensywne. Czasem daje to lepsze rezultaty uczenia, a czasem pogarsza wynik – warto jednak poznać to „na własnej skórze". Drugi parametr, opisany w okienku programu **Example 09** symbolem *Eta,* reprezentuje *współczynnik bezwładności uczenia.* Współczynnik ten, nazywany zwykle w literaturze *momentum*, powoduje, że sieć podczas uczenia silniej stara się zachować pewien wcześniej przyjęty kierunek zmian współczynników wagowych, przez co jest nieco „konserwatywna". I znowu – w niektórych przypadkach zwiększenie wartości tego współczynnika polepsza uczenie – a w innych przeszkadza.

Oba wskazane współczynniki stale są dostępne w oknie programu **Example 09**, możesz je w związku z tym zmieniać w trakcie uczenia, gromadząc bardzo ciekawe obserwacje. Najpierw możesz podjąć jednorazową (tzn. przed rozpoczęciem uczenia) próbę modyfikacji tych parametrów. Zauważysz wtedy głównie zwiększanie się szybkości uczenia wraz ze wzrostem współczynnika uczenia *Alpha*. Na przykład, na rysunku 8.33 widzisz przebieg procesu uczenia pewnego zadania przy standardowych wartościach współczynnika uczenia, a na rysunku 8.34 widzisz, jak ta sama sieć uczy się rozwiązywać to samo zadanie przy zwiększonej wartości tego współczynnika.

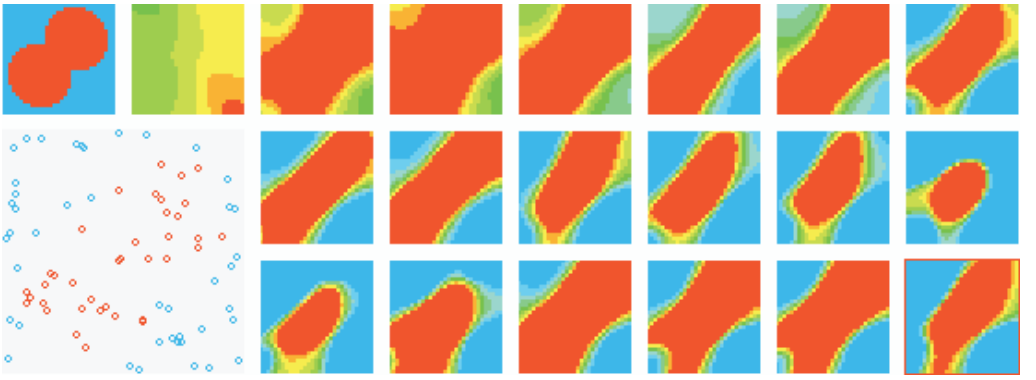


Rys. 8.33. Standardowa wartość współczynnika uczenia prowadzi do spokojnego i równomiernego uczenia, które jest jednak dość powolne

Zbyt duża wartość współczynnika uczenia jest jednak także niekorzystna, gdyż pojawiają się wtedy znamiona destabilizacji procesu uczenia, co dobitnie ilustruje rysunek 8.35. Sposobem na wygaszenie oscylacji sieci, występujących zawsze, gdy współczynnik uczenia wzrośnie nadmiernie, jest wpro-



Rys. 8.34. Zwiększona wartość współczynnika uczenia powoduje niekiedy, że proces uczenia przebiega szybciej

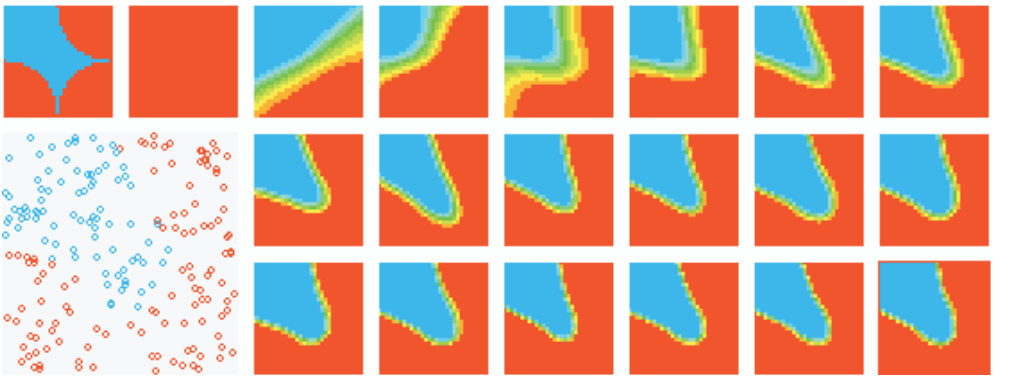


Rys. 8.35. Nadmierna wartość współczynnika uczenia powoduje, że proces uczenia cechują oscylacje – sieć na przemian zbliża się do pożądanego rozwiązania i od niego ucieka

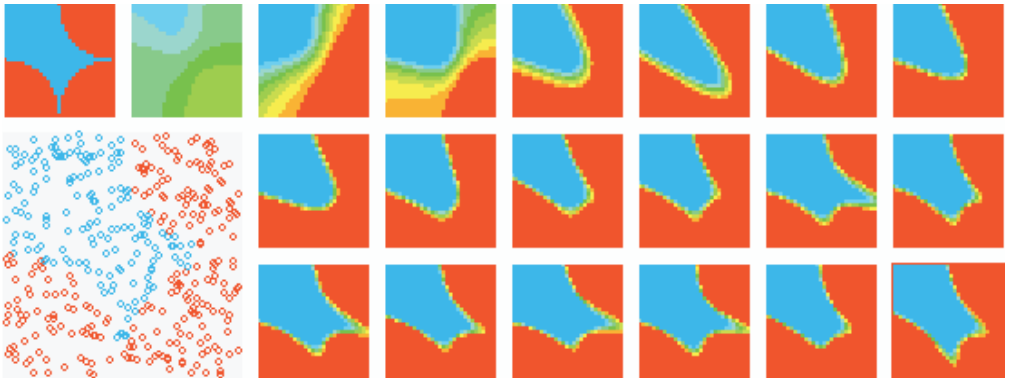
wadzenie większej wartości współczynnika *Eta* reprezentującego bezwładność tego procesu, czyli *momentum.* Mógłbyś więc, gdy zechcesz, przebadać wpływ zmian wartości współczynnika *momentum* na uczenie sieci, a także będziesz mógł prześledzić stabilizujący wpływ parametru *momentum,* gdy zdarzy się, że proces uczenia wymknie się spod kontroli.

Innym obszarem bardzo ciekawych badań jest kwestia wpływu struktury sieci na jej zachowanie. W tym celu musisz wybrać jakieś naprawdę skomplikowane zadanie. Niezłe jest do tego celu zadanie pokazane na rysunkach 8.4 i 8.5. Komplikacja tego zadania polega na konieczności „wpasowania" się sieci w wąskie i ciasne „zatoki" tworzone przez błękitny obszar negatywnych decyzji. Każda taka szczelina wymaga od sieci bardzo precyzyjnego doboru

jej elementów do charakterystycznych cech rozważanego zadania i sprawia naprawdę duże trudności. Dlatego prosta sieć dwuwarstwowa nie ma tu żadnych szans – choćbyś ją nie wiem jak długo uczył, zawsze będzie miała tendencje do upraszczania kształtu rozpoznawanych obszarów (rys. 8.36).



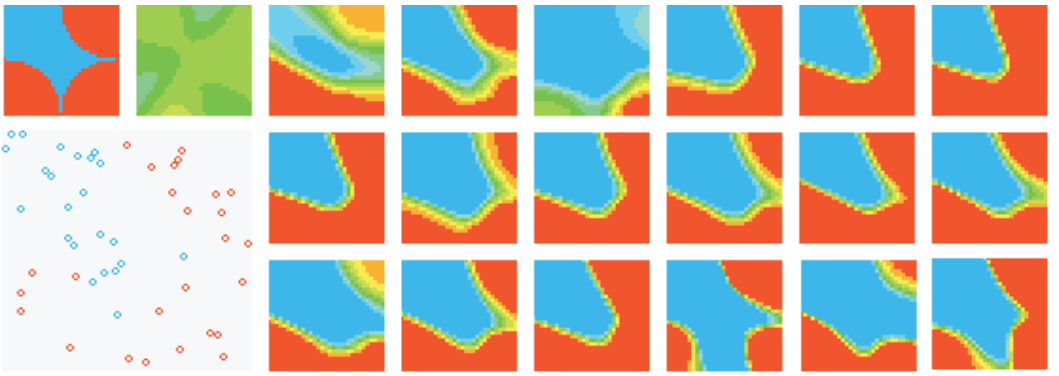
Rys. 8.36. Słabe dopasowanie obszarów w przypadku sieci dwuwarstwowej



Rys. 8.37. Poprawne i szybkie rozwiązanie trudnego zadania przy użyciu sieci trójwarstwowej

Natomiast sieć trójwarstwowa potrafi wydobyć wszystkie subtelności rozważanego zadania (rys. 8.37), chociaż niekiedy trudniej jest uzyskać stabilny przebieg procesu jej uczenia, bo sieć – będąc nawet bardzo blisko poprawnego rozwiązania, „ucieka" od niego pod wpływem wstecznej propagacji błędów i „myszkuje" nie znajdując w efekcie wcale poprawnego rozwiązania (rys. 8.38).

Oczywiście, teraz możesz puścić wodze fantazji i możesz zupełnie dowolnie rozmieścić obszary, w których sieć powinna podejmować „pozytyw-



Rys. 8.38. Znamiona niestabilności uczenia pojawiające się w sieci trójwarstwowej

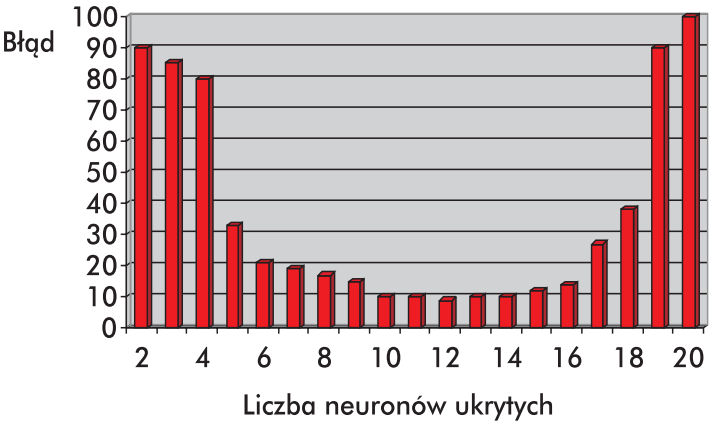
ne" i „negatywne" decyzje. Możesz przeprowadzić dziesiątki badań i uzyskać setki obserwacji. Jednak uprzedzam: weź pod uwagę czynnik czasu: uczenie dużej sieci będzie postępowało naprzód bardzo wolno, dlatego potrzebować będziesz **albo** bardzo szybkiego komputera, **albo** bardzo wiele cierpliwości. Nie chcę Ci sugerować wyboru, dla porządku jednak odnotuję, że cierpliwość jest tańsza.

## 8.7. Pytania kontrolne i zadania do samodzielnego rozwiązania

1. Czy problematyka rozpoznawania obrazów dotyczy tylko obrazów jako takich (na przykład z kamery cyfrowej), czy też ma szersze zastosowania?
2. W jakiej formie wprowadza się obrazy (oraz informacje o innych rozpoznawanych obiektach) do sieci neuronowej pracującej w roli klasyfikatora?
3. W jaki sposób sieci rozpoznające przekazują (najczęściej) użytkownikowi wyniki swojej pracy, czyli informację o tym, co zostało rozpoznane? Jakie wynikają z tego wnioski na temat typowej struktury warstwy wyjściowej tych sieci?
4. Spróbuj skonfrontować wnioski wynikające z obserwacji opisywanego w tym rozdziale uczenia sieci i jej działania w różnych generowanych przez program zadaniach – z teoretycznymi informacjami na temat zachowania sieci o różnej liczbie warstw, przedstawionymi w rozdziale 6. Zauważ bogatsze możliwości zachowania opisywanych w tym rozdziale sieci, wynikające z faktu, że neurony w tych sieciach mogą produkować dowolne **pośrednie** sygnały wyjściowe, a nie tylko **0** oraz **1** (co odpowiada niebieskiemu i czerwonemu kolorowi na rysunkach produkowanych przez program).

5. Zastanów się, jaki jest związek pomiędzy liczbą neuronów w złożonej wielowarstwowej sieci (w jej warstwach ukrytych) a liczbą oddzielnych obszarów w przestrzeni sygnałów wejściowych, w których sieci te po nauczeniu dają przeciwstawne decyzje (**0** oraz **1**).

6. Zależność pomiędzy liczbą neuronów w warstwie ukrytej a sprawnością wykonywania przez sieć określonych (wyuczonych) zadań przedstawiona jest przykładowo na rysunku 8.39. Lewą stronę tego wykresu stosunkowo łatwo zinterpretować: oto sieć mająca zbyt mało neuronów ukrytych jest zbyt mało „inteligentna", żeby poprawnie rozwiązać postawiony problem, a zwiększenie liczby tych neuronów powoduje polepszenie działania sieci. Jak jednak zinterpretować fakt (też wielokrotnie dowiedziony w sieci neuronowej), że nadmierne zwiększenie liczby neuronów ukrytych także – paradoksalnie – prowadzi do pogorszenia jakości działania sieci?




Rys. 8.39. Ilustracja do zadania nr 6

7. Współczynnik uczenia (*learning rate*) jest miarą intensywności zmian wag dokonywanych w sieci po wykryciu jej błędnego działania. Jego potoczna interpretacja jest taka, że duże wartości tego współczynnika odpowiadają działaniom „surowego i wymagającego nauczyciela", a małe wartości tego współczynnika modelują „nauczyciela pobłażliwego i wyrozumiałego". Spróbuj uzasadnić fakt, że proces uczenia przebiega najskuteczniej przy korzystaniu z wartości współczynnika uczenia (*learning rate*) mających wartości umiarkowane (ani za duże, ani za małe). Spróbuj zebrać i uporządkować informacje na ten temat, eksperymentując z opisanym wyżej programem. Narysuj wykresy zmian jakości działania sieci po pewnej ustalonej liczbie kroków uczenia (na przykład po wykonaniu 500, 1000 i 5000 kroków), w zależności od wartości współczynnika uczenia.

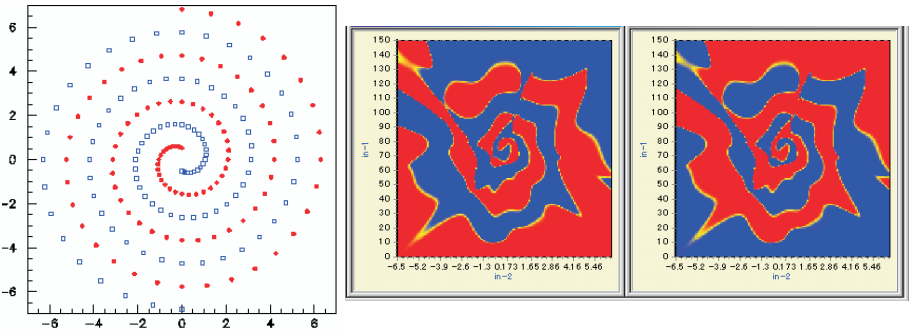
8. W technice sieci neuronowych rozważanych jest wiele algorytmów uczenia, w których wartości współczynnika uczenia zmieniają się w trakcie uczenia. Jak sądzisz, w jakich okolicznościach wartości te powinny być zwiększane, a w jakich zmniejszane?

9. Niektóre teorie uczenia sieci neuronowych zawierają zalecenie, żeby współczynnik uczenia na początku procesu uczenia miał małą wartość, bo sieć wtedy popełnia dużą liczbę poważnych błędów, i gdyby proporcjonalnie do tych dużych błędów silnie zmieniać wartości wag – to sieć „miotałaby się konwulsyjnie", unikając jednych błędów, ale na skutek zbyt dużych zmian popadałaby w inne błędy (dla innych przykładów uczących). Na początku potrzebna jest więc sieci „łagodna i delikatna przedszkolanka". Potem, w miarę gromadzenia przez sieć coraz większej liczby umiejętności, można współczynnik uczenia zwiększyć, bo sieć popełnia mniej błędów i są one niewielkie, więc dla skutecznego postępu procesu uczenia trzeba stosować „dokręcanie śruby". Natomiast pod koniec procesu uczenia, gdy sieć zgromadziła już dużą ilość wiedzy – znowu zalecane jest zmniejszenie wartości współczynnika uczenia, żeby przypadkowe, incydentalne błędy (na przykład związane ze szczególnie trudnymi lub nietypowymi obiektami podlegającymi klasyfikacji) nie psuły całościowego efektu (bo poprawienie działania sieci w jednym punkcie zawsze powoduje jakieś – nawet minimalne – popsucie jej działania w innych punktach). Spróbuj przeprowadzić serię eksperymentów potwierdzających tę teorię – lub wykazujących jej niesłuszność.

10. Skutki nadmiernej wartości współczynnika uczenia (zagrażającej stabilności uczenia) kompensuje się zwiększeniem współczynnika *momentum*, będącego miarą stopnia „konserwatyzmu" w procesie uczenia. Zaplanuj i przeprowadź serię badań pozwalających na narysowanie wykresu ukazującego, jakie są wymagane minimalne wartości współczynnika *momentum* dla różnych wartości współczynnika uczenia, które łącznie gwarantowałyby najlepszy (szybki, ale stabilny) przebieg procesu uczenia.

11. **Zadanie dla zaawansowanych:** Zmodyfikuj program używany w badaniach uczenia sieci w taki sposób, żeby możliwe było zadawanie obszarów przynależności obiektów do poszczególnych klas (obszarów, w których „zwierzak" jest szczęśliwy i nieszczęśliwy) poprzez rysowanie ich granic na ekranie za pomocą edytora graficznego. Korzystając z takiego narzędzia, spróbuj zbudować sieć, która zdoła się nauczyć zadania rozpoznawania szczególnie trudnego problemu, w którym obszary pozytywnych i negatywnych decyzji sieci układają się w formie dwóch spirali (rys. 8.40).

12. **Zadanie dla zaawansowanych:** Zaprojektuj i wykonaj model „zwierzaka", który będzie miał więcej receptorów („narządów zmysłów" dostarczających informacje o różnych cechach „środowiska") oraz będzie miał możli-




Rys. 8.40. Ilustracje do zadania nr 11. Problem „dwóch spirali": zbiór uczący (po lewej) i jego dwa przykładowe rozwiązania

wość bardziej złożonych działań (na przykład będzie się mógł przemieszczać w „środowisku" szukając określonych warunków, jakie mu odpowiadają, albo szukając jakichś obiektów – pożywienia itp.). Obserwacja działania uczącej się sieci neuronowej jako „mózgu" takiego zwierzaka oraz jego zachowań w środowisku może być fascynującą przygodą intelektualną!

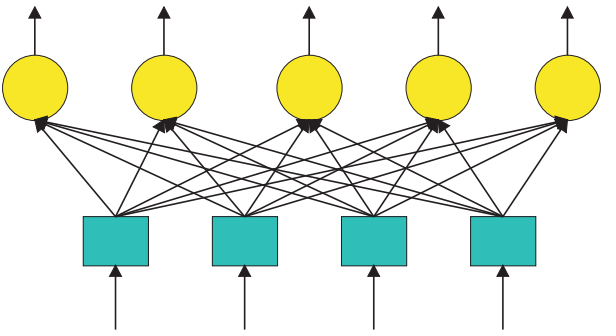
# 9. Sieci neuronowe samouczące się

## 9.1. Na czym polega idea samouczenia sieci?

Widziałeś już (i wypróbowałeś w kilku programach!), jak korzystając ze wskazówek nauczyciela sieć neuronowa z kroku na krok doskonaliła swoje działanie, osiągając po pewnym czasie zgodność swego zachowania z podanymi przez nauczyciela wzorami.

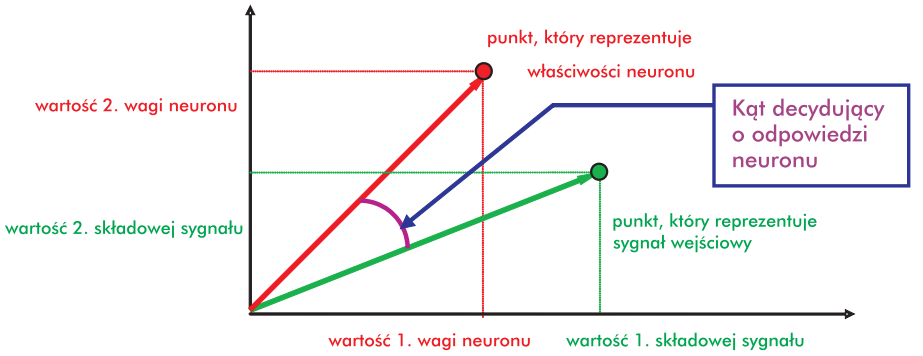
Teraz pokażę Ci, że sieć może się uczyć **całkiem sama,** bez nauczyciela. Pisałem Ci już na ten temat w rozdziale 3.2, ale teraz przyszedł czas, żebyś poznał kilka konkretów. Musisz wiedzieć, że istnieje kilka metod takiego samouczenia (algorytmy Hebba, Oji, Kohonena itd.), ale w tej książce nie wchodzimy w szczegóły algorytmów, tylko spróbujemy zobaczyć, na czym to wszystko polega. W tym celu proponuję, żebyś uruchomił program **Example 10a.**

W programie tym masz do czynienia z siecią neuronów, mającą postać jednej tylko, ale za to czasem dosyć obszernej warstwy (rys. 9.1). Do wszystkich neuronów tej warstwy podawane są te same sygnały wejściowe i wszystkie neurony wyznaczają (niezależnie od siebie) wartość swojego stopnia pobudzenia przemnażając sygnały wejściowe (identyczne dla wszystkich neuronów) przez swoje współczynniki wagowe (oczywiście inne w każdym neuronie).



Rys. 9.1. Struktura jednowarstwowej sieci, która może podlegać samouczeniu

Jak pamiętasz, pobudzenie neuronu jest więc tym silniejsze, im lepsza będzie zgodność między sygnałami wejściowymi podanymi w danej chwili a wewnętrznym wzorcem. Wiedząc, jakie neuron ma współczynniki wagowe, będziesz wiedział zatem, na które sygnały wejściowe będzie on reagował pozytywnie, a na które negatywnie. Jeśli założymy, że sygnały wejściowe są dwuwymiarowe (to znaczy mają dwie składowe) i możesz je zaznaczać jako punkty na płaszczyźnie (nazywamy ją przestrzenią sygnałów wejściowych, pamiętasz?), wtedy również wagi wszystkich neuronów też są dwuwymiarowe (wag musi być zawsze tyle samo, ile składowych sygnału) i też można je zaznaczać jako punkty na płaszczyźnie – najwygodniej **na tej samej płaszczyźnie**. W ten sposób na płaszczyźnie sygnałów wejściowych pojawią się punkty sygnalizujące lokalizacje poszczególnych neuronów, wynikające z ich wewnętrznej wiedzy, czyli z wartości przyjmowanych przez ich współczynniki wagowe. Przypominam Ci to na rysunku 9.2., który widziałeś już w jednym z wcześniejszych rozdziałów, ale teraz – bogatszy o tyle nowych doświadczeń – zapewne spojrzysz na niego zupełnie nowym okiem.




Rys. 9.2. Interpretacja wzajemnego położenia wektora wag i wektora sygnału wejściowego

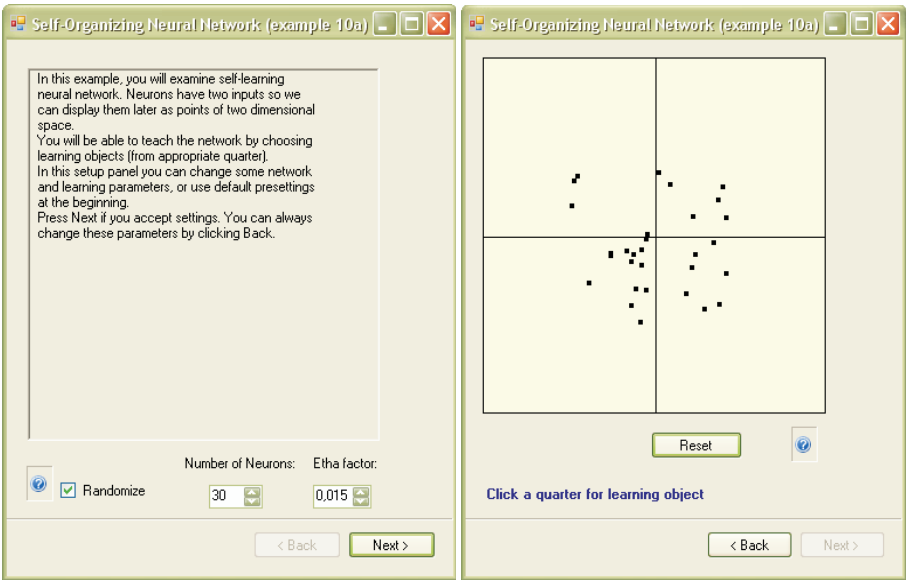
Zasada samouczenia polega na tym, że na początku wszystkie neurony otrzymują **przypadkowe** wartości współczynników wagowych, co na rysunku produkowanym przez program **Example 10a** zobaczysz jako „chmurkę" przypadkowo rozrzuconych punktów. Każdy z tych punktów reprezentuje jeden neuron; jak zwykle położenie punktów wynika z wartości współczynników wag. Przedstawia Ci to rysunek 9.3, ekran po prawej stronie. Jeśli chodzi o ekran po lewej stronie rys. 9.3, to jak w poprzednich programach pokaże się on zaraz po uruchomieniu programu i będzie służył Ci do modyfikacji parametrów sieci. Parametry do ewentualnego ustawienia masz dwa: liczbę neuronów w sieci (więcej neuronów potrafi się ciekawiej zachowywać, ale trudniej jest to śledzić) oraz parametr nazwany ***Etha***, który reprezentuje intensywność

procesu uczenia (większym wartościom odpowiadać będzie szybsza i bardziej energiczna nauka). Możesz te parametry w każdej chwili zmienić, na początku jednak proponuję Ci zaakceptować proponowane przez program wartości. Te domyślne ustawienia zostały wybrane przeze mnie na podstawie wstępnego przebadania zachowania programu. Do opisu poszczególnych parametrów i do „majstrowania" przy ich wartościach powrócimy w dalszej części rozdziału.

Po uruchomieniu programu będziesz miał sieć złożoną z 30 neuronów, cechującą się raczej umiarkowanym zapałem do nauki. Wszystkie wagi wszystkich neuronów w tej sieci będą miały wartości przypadkowe. Do takiej przypadkowo zainicjowanej sieci zaczynają napływać sygnały wejściowe, które stanowią przykładowe próbki obiektów, z którymi sieć ma do czynienia (pamiętasz może z drugiego rozdziału przykład, w którym występowali Marsjanie, Marsjanki i Marsjanięta?). Na pojawianie się tych obiektów w programie **Example 10a** będziesz miał wpływ. Kłóci się to trochę z ideą samouczenia (w końcu jak sieć ma się sama uczyć, to Ty nie powinieneś mieć nic do roboty!), jednak dopiero w dalszych doświadczeniach proces prezentacji obiektów ciągu uczącego (bez żadnych informacji na temat tego, co z nimi trzeba robić!) będzie przebiegał – tak jak w rzeczywistych zastosowaniach – niezależnie od Ciebie, w sposób losowy. W programie **Example 10a** zrobiłem celowo odstępstwo od tego ideału, ponieważ chciałbym, żebyś mógł sam zaobserwować, jak sposób pojawiania się obiektów uczących (które częściej, które rzadziej...) wpływa na proces samodzielnego „zdobywania wiedzy" przez sieć.

Konieczne jest tu jednak pewne zastrzeżenie. Otóż podczas samouczenia cała wiedza, jaką sieć może zdobyć, zawierać się musi w pokazywanych obiektach wejściowych, a dokładniej w tym, że obiekty te tworzą pewne **klasy podobieństwa**, to znaczy są wśród nich takie, które **są do siebie nawzajem podobne** (***Marsjanie***) i **są odmienne od tych, które należą do innej klasy** (***Marsjanki***). Oznacza to, że ani Ty, podając sieci obiekty uczące „z palca", ani automatyczny generator, który zastąpi Cię w tej pracy w następnym programie – **nie możecie** pokazywać sieci obiektów o **całkiem przypadkowym** położeniu w przestrzeni sygnałów wejściowych. Pokazywane obiekty powinny tworzyć wyraźne skupiska wokół pewnych, dających się wykryć ośrodków, którym będzie można nadać nazwy oraz interpretacje („typowy Marsjanin", „typowa Marsjanka" itd.).

Podczas eksploatacji programu **Example 10a** do Ciebie każdorazowo należeć będzie decyzja, co się teraz sieci pokaże, ale będzie to wyłącznie bardzo „zgrubny" wybór pokazywanego obiektu, bo jego szczegóły ustali za Ciebie automat. Konkretnie będzie Ci wolno wskazać, w obrębie której ćwiart-

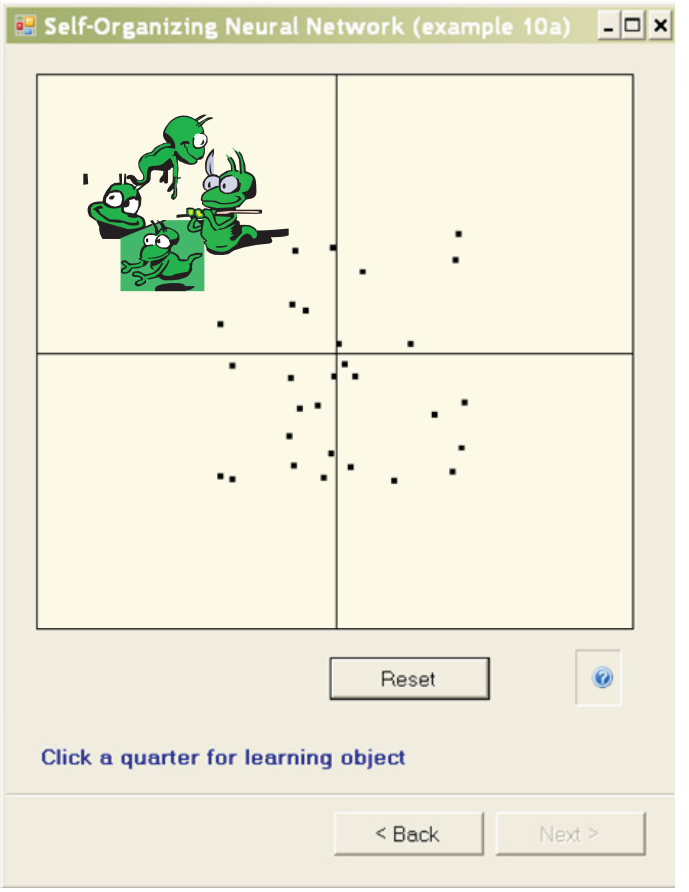



Rys. 9.3. Początkowe okno z parametrami (po lewej) oraz położenie punktów reprezentujących neurony w programie **Example 10a** po zaakceptowaniu parametrów sieci i wybraniu *Next* (okno po prawej stronie)

ki[1] znajdować się ma kolejny pokazywany obiekt. Wystarczy, że w tym celu klikniesz myszką na odpowiedniej ćwiartce. W ten sposób będziesz sterował tym, co sieć „zobaczy". Możesz się na przykład umówić (sam ze sobą!), że obiekty pokazywane w pierwszej ćwiartce to ***Marsjanie***, w drugiej ***Mar-***

[1] Pojęcie „ćwiartki" (jakiejś przestrzeni, zdefiniowanej w oparciu o dwuwymiarowy układ współrzędnych) wywodzi się z matematyki, a ja obiecywałem, że przy czytaniu tej książki wiedza matematyczna nie będzie wymagana. Zatem jeśli nie uczyłeś się jeszcze w matematyce (albo uczyłeś się i zapomniałeś), co to jest układ współrzędnych i jakie ćwiartki wyznacza on w przestrzeni opisywanych w nim sygnałów, to przyjmij do wiadomości, że ćwiartki numeruje się poczynając od prawego górnego narożnika rozważanej przestrzeni. Ta prawa górna ćwiartka, w której zmienne odkładane na obu osiach przyjmują wyłącznie wartości dodatnie, numerowana jest jako **pierwsza.** Natomiast kolejne następne ćwiartki numerowane są kolejnymi numerami przy obieganiu układu współrzędnych **w kierunku przeciwnym do ruchu wskazówek zegara.** Zatem ćwiartka druga to ta w lewym górnym rogu, ćwiartka trzecia jest zaraz pod nią itd. Jeśli chciałbyś zobaczyć sposób rozmieszczenia ćwiartek (wraz z ich numerami) na rysunku – to przewróć kilka kartek do przodu i znajdź rysunek 9.13. Na rysunku tym w prawym dolnym rogu prezentowanego tam okna programu widzisz układ czterech przycisków opisanych stosownymi numerami, odpowiadający dokładnie opisanemu tu układowi czterech ćwiartek. O tym, co robią te przyciski, dowiesz się później – ale teraz przyjrzyj się, jak są one rozmieszczone oraz zapamiętaj, gdzie są poszczególne numery. Ułatwi Ci to śledzenie dalszej treści rozdziału.

***sjanki***, w trzeciej ***Marsjanięta***, a w czwartej – ***Marsjamory***[2]. Przy odrobinie wyobraźni możesz spojrzeć na okienko prezentowane przez program **Example 10a** w taki sposób, jak to pokazałem na rysunku 9.4, gdzie spróbowałem w układzie współrzędnych zamiast punktów reprezentujących **pomierzone cechy** Marsjan – pokazać Ci Marsjan jako takich.




Rys. 9.4. Reprezentacja punktów przestrzeni sygnałów wejściowych jako „Marsjan"

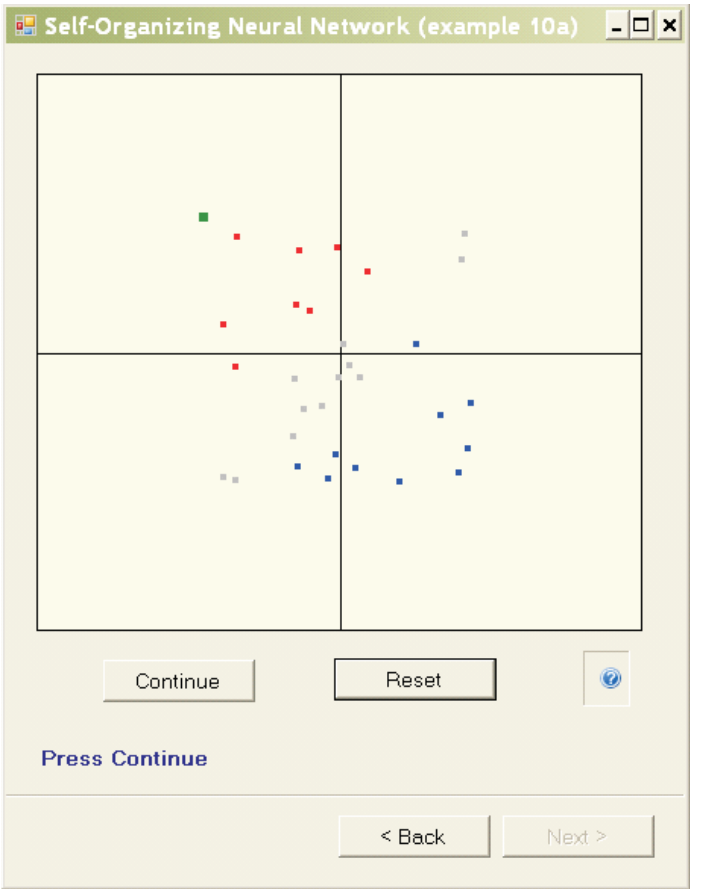
Jak widzisz na rysunku – każdy Marsjanin jest trochę inny, ale są oni do siebie nawzajem podobni i dlatego ich dane są reprezentowane w tym samym

[2] Marsjamory są to przedstawiciele obecnej na Marsie trzeciej płci. Jak ogólnie wiadomo, mieszkańcy Marsa celem spłodzenia potomka łączą się w trójkąty, co zresztą powoduje, że zajęci swoim skomplikowanym życiem erotycznym nie wytworzyli jeszcze cywilizacji technicznej, nie tworzą dużych budowli i dlatego obserwacje prowadzone z Ziemi nie są w stanie ich wykryć.

obszarze przestrzeni sygnałów wejściowych (co ułatwi sieci neuronowej wyuczenie się tego, jak wygląda „typowy Marsjanin"). Jednak ponieważ każdy z tych Marsjan ma swoje indywidualne cechy odróżniające go trochę od pozostałych Marsjan, dlatego punkty reprezentujące ich podczas procesu samouczenia będą się nieco różniły od siebie – to znaczy będą lokowały się w nieco innym miejscu naszego układu współrzędnych. Ponieważ jednak więcej jest cech, które powodują, że Marsjanie różnią się między sobą, niż tych cech, które odróżniają każdego z nich indywidualnie – punkty ich reprezentujące będą się skupiały w pewnym konkretnym podobszarze rozważanej przestrzeni sygnałów wejściowych.

Generując obiekty do procesu uczenia będziesz mógł zdecydować, czy teraz ma być pokazany ***Marsjanin, Marsjanka, Marsjaniątko*** czy ***Marsjamor.*** Nie będziesz natomiast podawał żadnych szczegółowych danych każdego konkretnego obiektu (tzn. nie musisz określać np. wzrostu ani koloru oczu każdego Marsjanina), ponieważ to zrobi już program automatycznie. Program na każdym kroku procesu samouczenia zapyta Cię, z jakim okazem Marsjanina masz do czynienia – to znaczy z której ćwiartki ma pochodzić pokazany sieci obiekt. Gdy się zdecydujesz i wybierzesz którąś ćwiartkę – program wygeneruje obiekt określonej klasy i „pokaże" go sieci. Każdy pokazany Marsjanin będzie trochę inny, ale łatwo zauważysz, że ich cechy (wyrażające się lokalizacją odpowiadających im punktów w przestrzeni wejściowych sygnałów) będą się wyraźnie grupowały wokół pewnych „prototypów". Być może ten proces „wyczarowywania" i pokazywania sieci poszczególnych obiektów wydaje Ci się w tej chwili trochę skomplikowany, ale po krótkiej chwili pracy z programem **Example 10a** nabierzesz wprawy, a ponadto w programie ukazywać się będą podpowiedzi, pomagające Ci zorientować się, o co w danej chwili chodzi.

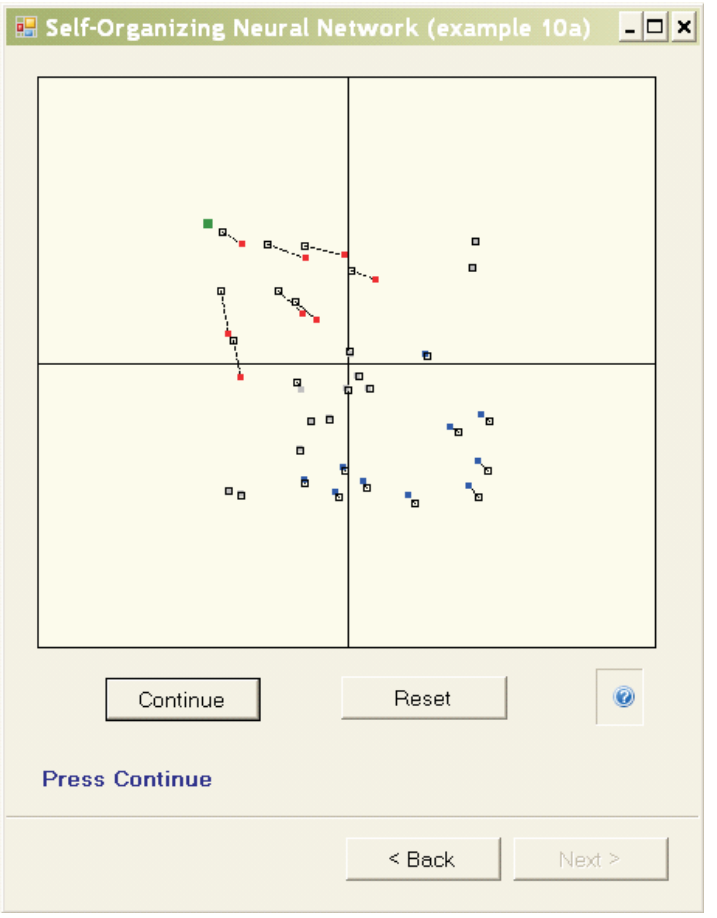
W momencie pojawienia się określonego sygnału wejściowego, należącego do ciągu uczącego, zobaczysz go na ekranie jako zielony kwadrat (spróbuj zgadnąć, dlaczego wybrałem właśnie kolor **zielony**), zdecydowanie większy od punktu oznaczającego położenie każdego dowolnego neuronu (rys 9.5). Otrzymawszy ten sygnał na swoich wejściach – wszystkie neurony samouczącej się sieci określą (na podstawie składowych tego sygnału wejściowego oraz swoich wag) swoje sygnały wyjściowe. Sygnały te mogą być dodatnie (dany obiekt się temu neuronowi „podoba" – punkty takie zobaczysz na ekranie jako czerwone) albo ujemne (dany obiekt się temu neuronowi „nie podoba" – punkty takie zobaczysz na ekranie jako niebieskie). Jeśli sygnały wyjściowe pewnych neuronów (zarówno dodatnie, jak i ujemne) mają małą wartość – odpowiednie punkty przyjmują barwę szarą (odpowiednie neurony „mają do pojawiającego się osobnika obojętny stosunek").




Rys. 9.5. Sytuacja po pojawieniu się obiektu uczącego („Marsjanina”)

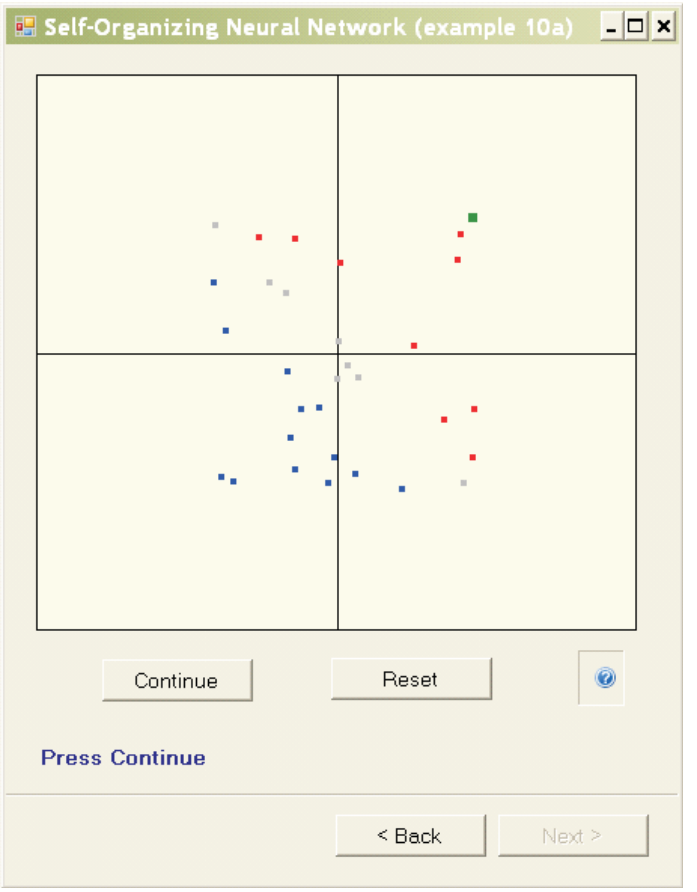
Na podstawie sygnałów wejściowych i ustalonych wyjść wszystkie neurony samouczącej się sieci korygują swoje wagi. Podczas korekty wag zachowanie każdego neuronu zależy od tego, jaka była wartość jego sygnału wyjściowego, którym odpowiedział on na pobudzenie. Jeśli sygnał wyjściowy neuronu był silnie pozytywny (na ekranie – czerwony) – wagi zmieniają się w taki sposób, że neuron zbliża się do punktu, który mu się tak podobał. Oznacza to, że jeśli ponownie zostanie pokazany ten sam punkt – neuron odnotuje go jeszcze bardziej entuzjastycznie (sygnał wyjściowy będzie miał jeszcze większą dodatnią wartość). Z kolei neurony, które wykazywały w stosunku do aktualnie pokazywanego wzorca stosunek negatywny (na ekranie widoczne jako punkty niebieskie) – będą od wzorca „odpychane“, w wyniku czego ich wagi skłonią je w przyszłości do jeszcze bardziej negatywnego reagowania na

ten typ wejścia. W programie **Example 10a** można zauważyć efekt „ucieczki" negatywnie nastawionych neuronów poza obszar wyświetlania punktów, które je reprezentują. W tym przypadku neuron przestaje być wyświetlany. Jest to naturalny efekt, jeśli weźmie się pod uwagę fakt, że w procesie samouczenia sieci neuronowej zazwyczaj stosuje się równie silne „przyciąganie", jak i „odpychanie".




Rys. 9.6. Korekta wag wywołana samouczeniem

Procesy przesuwania się punktów pokazujących położenia wektorów wag neuronów możesz bardzo dokładnie obejrzeć, ponieważ program prezentuje w każdym kolejnym kroku obraz, na którym widoczne są zarówno poprzednie (kolorowe), jak i nowe (kwadraty, puste w środku) położenia punktów reprezentujących neurony – rys. 9.6, co więcej – stare położenie jest z nowym

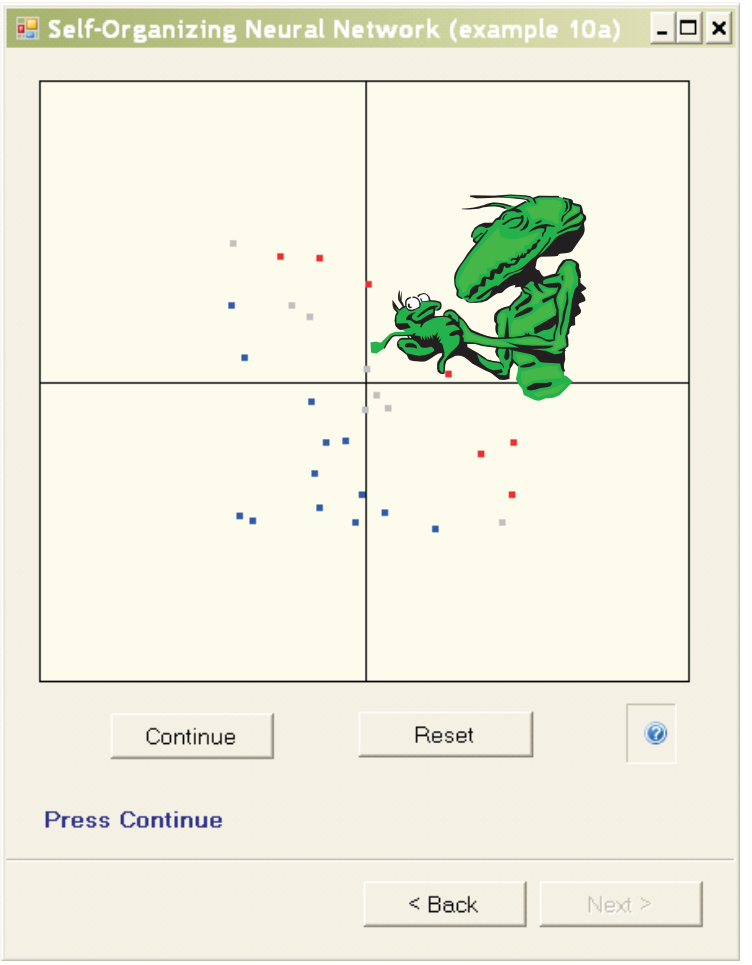



Rys. 9.7. Kolejna iteracja procesu samouczenia

połączone kropkowaną linią, można więc śledzić „drogę", jaką przebywają uczące się neurony.

W kolejnym kroku „nowe" wagi neuronów stają się „starymi" i po pokazaniu kolejnego punktu cały cykl zaczyna się od początku (rys. 9.7), a sieci pokazany zostaje nowy obiekt, być może należący do nowej klasy (rys. 9.8).

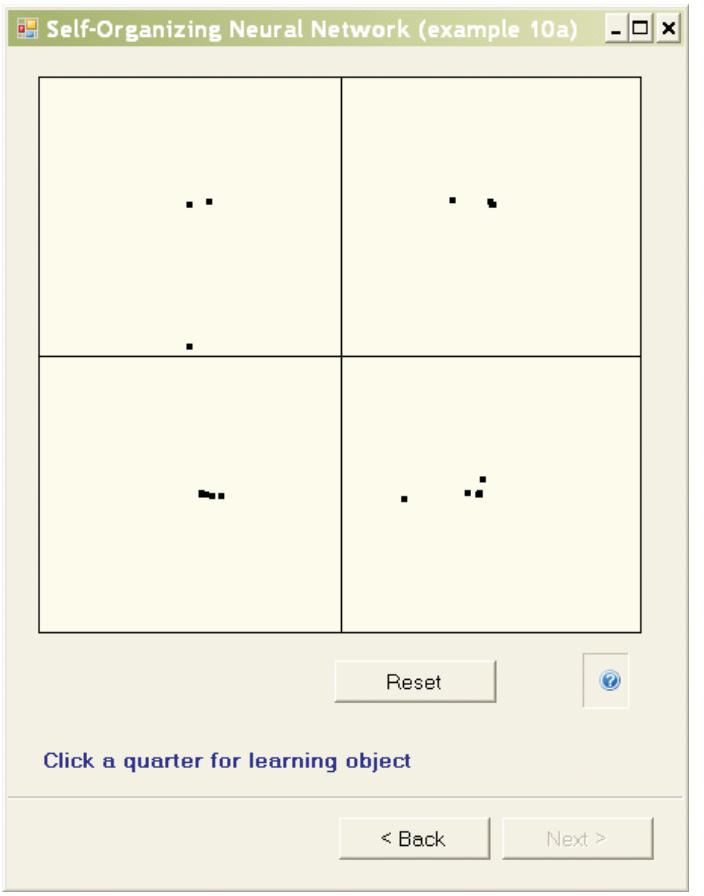
Jeśli będziesz ten proces dostatecznie długo powtarzał – powstanie w każdej ćwiartce skupisko neuronów, które będą wyspecjalizowane w rozpoznawaniu typowego obiektu należącego do tej właśnie grupy (rys. 9.9). W ten sposób sieć – całkiem sama – wyspecjalizuje się w rozpoznawaniu wszystkich napotkanych rodzajów obiektów. Śledząc proces samouczenia zauważysz na pewno, że utrwala on i pogłębia jedynie „wrodzone skłonności" sieci neuronowej, wyrażające się w początkowym (losowym!) rozrzucie wektorów




Rys. 9.8. Interpretacja kolejnej iteracji procesu samouczenia (z rysunku 9.7)

wag. Te właśnie początkowe różnice powodują, że po pojawieniu się obiektu wejściowego jedne neurony reagują na jego pokazanie pozytywnie, a inne negatywnie, a potem następuje korekta wag wszystkich neuronów sieci w taki sposób, że w rezultacie uczy się ona rozpoznawać typowe pojawiające się na wejściu elementy i skutecznie doskonali swoją zdolność rozpoznawania **bez żadnej ingerencji nauczyciela,** który może nawet nie wiedzieć, ile obiektów trzeba będzie rozpoznawać ani jakie będą ich cechy.

Do tego, aby proces samouczenia przebiegał efektywnie, konieczne jest jednak zapewnienie w populacji neuronów niezbędnej początkowej różnorodności. Jeśli wartości wag są losowane w taki sposób, by w populacji neuronów



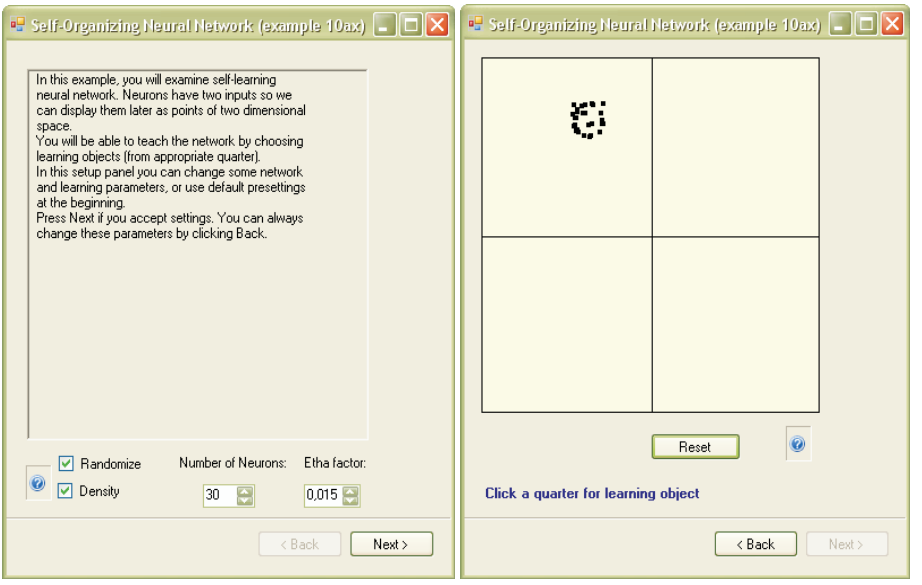

Rys. 9.9. Końcowa faza procesu samouczenia

**była** zapewniona niezbędna różnorodność – proces samouczenia przebiegać będzie stosunkowo łatwo i bez zakłóceń. Jeśli jednak wszystkie neurony będą miały podobne „wrodzone zamiłowania" – bardzo trudno będzie spowodować, by w toku procesu uczenia „obsłużyły" wszystkie klasy pojawiających się obiektów. Bardziej prawdopodobne jest w takim przypadku zjawisko „owczego pędu" – zbiorowej fascynacji wszystkich neuronów jednym napotkanym obiektem i całkowitego ignorowania wszystkich innych. Koniec najczęściej jest taki, że wszystkie neurony wyspecjalizują się w rozpoznawaniu jednej wybranej klasy obiektów – przy całkowitym ignorowaniu wszystkich pozostałych klas.

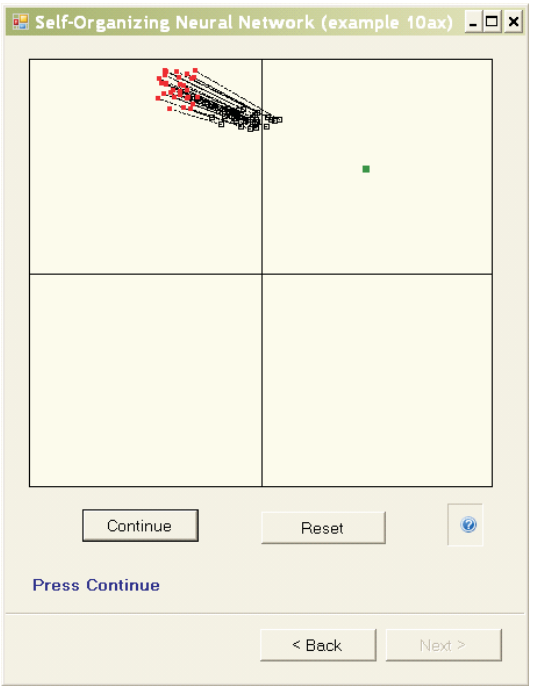
W programie **Example 10a** różnorodność była zagwarantowana jako coś oczywistego i niezmiennego. Ponieważ jednak problem wpływu losowości

wstępnego rozsiania neuronów jest sprawą o podstawowym znaczeniu – przygotowałem wariant tego programu o nazwie **Example 10ax,** w którym musisz wyraźnie zażądać albo losowego rozrzucenia cech (szeroko po całej przestrzeni wejść), albo tego, aby początkowe położenia neuronów były wylosowane z pewnego wąskiego obszaru. W tym celu należy odpowiednio ustawić stan pola (*checkbox*) o nazwie **Density** w panelu początkowym programu (lewy ekran na rys. 9.10). Na początku lepiej będzie wybierać neurony szeroko rozrzucone i dlatego powinieneś pozostawić pole **Density** puste. W momencie kiedy jednak poznasz już podstawowe prawidłowości procesu samouczenia sieci i będziesz gotowy do tego, by zobaczyć coś istotnie nowego – zaznacz to pole. Zobaczysz wtedy na własne oczy, jak ważne dla sieci neuronowej bywa początkowe losowe rozrzucenie parametrów (wag) neuronów. Kilka obrazów prezentujących sposób samouczenia sieci w przypadku, gdy wartości początkowe wag neuronów są zbyt słabo zróżnicowane – pokazałem na rysunkach 9.10, 9.11 i 9.12.

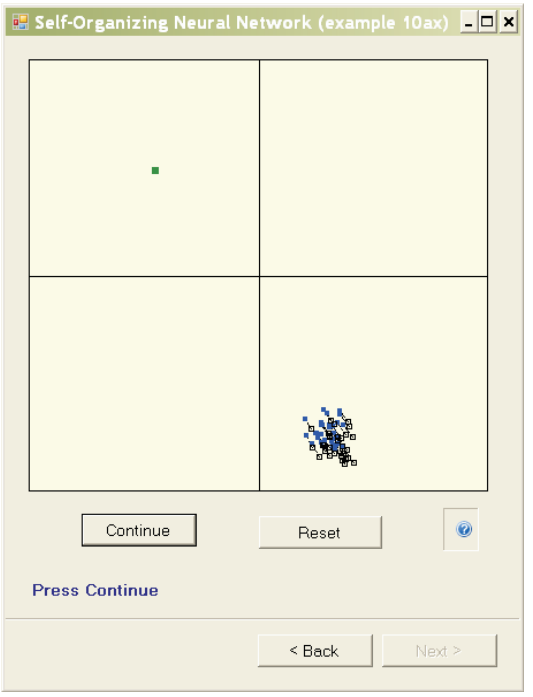
Możesz korzystając z programów **Example 10a** i **Example 10ax** przeprowadzić serię specjalistycznych badań. Na przykład możesz zbadać skutki




Rys. 9.10. Początkowe okno z parametrami (po lewej) oraz przykładowy efekt losowania początkowych wartości wektorów wag w programie **Example 10ax** po zaakceptowaniu parametrów sieci, w szczególności faktu zaznaczenia pola *Density.* W takim przypadku po kliknięciu przycisku *Next* dostajemy zadanie, w którym wartość początkowego rozrzutu wag neuronów jest za mała (okno po prawej stronie)




Rys. 9.11. Samouczenie w programie **Example 10ax** – wszystkie neurony zgodnie przyciągane przez ten sam atraktor




Rys. 9.12. Samouczenie w programie **Example 10ax** – wszystkie neurony zgodnie odpychane przez ten sam atraktor

bardziej „energicznego" samouczenia, które zobaczysz zmieniając (zwłaszcza zwiększając – ale ostrożnie!) wartość współczynnika uczenia – **Etha factor**.

Inne interesujące zjawiska możesz zaobserwować, odznaczając (to znaczy kasując początkowo istniejące zaznaczenie) parametru **Randomize**. Wtedy za każdym razem sieć będzie startowała z takim samym początkowym rozkładem współczynników wagowych, co spowoduje, że będziesz mógł zobaczyć, jak na końcowy rozkład położeń neuronów wpływa (zależna od twoich decyzji!) kolejność i częstość pokazywania obiektów z różnych ćwiartek. Kolejnym parametrem, z którym możesz eksperymentować, jest ilość neuronów (**Number of Neurons**), a co za tym idzie, także ilość rysowanych punktów. Parametry te, jak zapewne zdążyłeś się zorientować, można zmieniać wracając do okna ukazującego się zawsze bezpośrednio po uruchomieniu omawianych programów (rys. 9.3 i rys. 9.10 – początkowe okno z parametrami, pokazane po lewej stronie odpowiedniego rysunku).

Możesz na tym tle prowadzić bardzo ciekawe rozważania, odnajdując w początkowym rozkładzie położenia poszczególnych neuronów analogię do wrodzonych właściwości i talentów, a w sekwencji pokazywanych obiektów – analogię do indywidualnego doświadczenia życiowego osobnika. Mógłbym Ci wiele na ten temat opowiedzieć na podstawie tych setek godzin, jakie spędziłem przy modelowaniu różnych sieci neuronowych – są to jednak rozważania znacznie wykraczające poza obszar tej książki. Może kiedyś napiszę na ten temat wspomnienia?

## 9.2. Jak przebiega dłuższe samouczenie sieci?

Omówiony wyżej program **Example 10a** pozwalał bardzo dokładnie zobaczyć przebieg procesu samouczenia sieci neuronowej, był jednak niewygodny wtedy, gdy chciałeś zobaczyć, „czym się to wszystko skończy", bo trzeba wtedy było bardzo wiele razy naciskać przycisk **Continue** – co było męczące. Dlatego opracowałem wersję tego programu (**Example 10b**), która nie wymaga takiego wysiłku, ponieważ podstawowy proces samouczenia przebiega w niej „prawie" automatycznie. Analogicznie jak w poprzednich przykładach po uruchomieniu programu, akceptacji parametrów domyślnych lub po ich zmianie przechodzisz do okna, gdzie możesz obserwować eksperymenty. Proces samouczenia przebiega „prawie" automatycznie, ponieważ klikając i zwalniając przycisk **Start** możesz obserwować na ekranie proces samouczenia „krok po kroku" (żebyś się mógł przyjrzeć, co się dzieje) – a potem jeśli klikniesz raz na przycisk **Start** i nie zwolnisz kliknięcia – proces samouczenia stanie się automatyczny. W każdym momencie możesz powrócić do trybu „krok po kroku" albo ponownie „przyspieszyć" proces, stosując opisane wyżej podejście.

W ten sposób, nie wkładając żadnego wysiłku, możesz obserwować przebieg procesu samouczenia przez długi okres, dostrzegając szereg ciekawych zjawisk, o których za chwilę porozmawiamy.

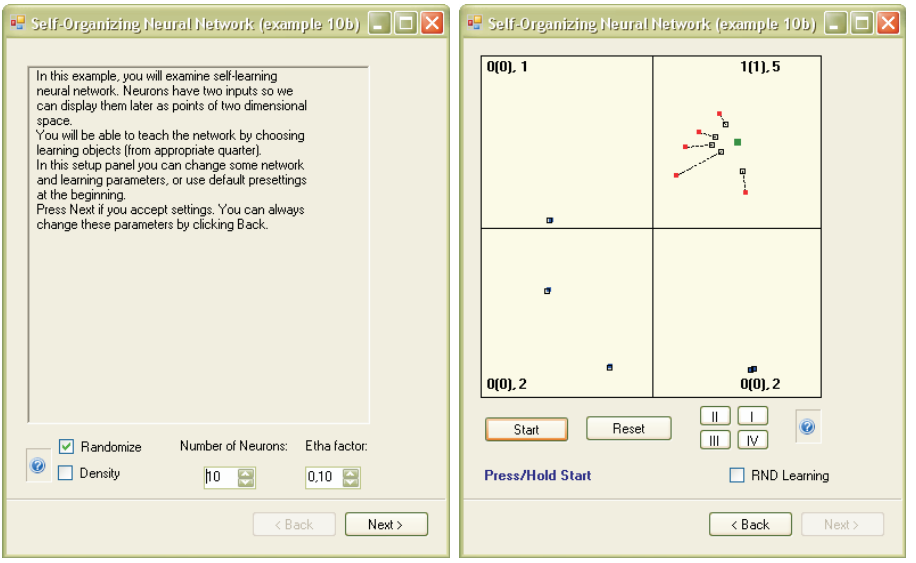
Kwestią, którą powinieneś rozważyć uruchamiając program w omawianym tu trybie, jest dobór rozmiaru sieci, jaką badasz. Program pozwala symulować sieć o swobodnie wybieranej liczbie neuronów. Na początku zacznij od obserwacji sieci złożonej z kilku neuronów (rys. 9.13, 9.14). Biorąc pod uwagę fakt, że sieć dążyć będzie do zbudowania czterech skupisk odpowiednio w czterech ćwiartkach – nie bardzo jest sens badania sieci zawierającej mniej niż cztery neurony, jednak jeśli chcesz – możesz obejrzeć nawet zachowanie **pojedynczego** neuronu. Całość symulacji program bierze na siebie, do Ciebie natomiast należy śledzenie przebiegu procesu samouczenia i wyciąganie wniosków. Dla ułatwienia obserwacji zachodzących w sieci procesów w każdej ćwiartce w rogu wyświetlane są dodatkowe informacje w postaci trzech liczb. Informują one (w kolejności występowania odpowiednich liczb), ile razy pokazano obiekt ciągu uczącego ulokowany w obrębie danej ćwiartki – w ciągu całego uczenia oraz (ze względu na dominującą rolę początkowych pokazów) w ciągu pierwszych dziewięciu prezentacji (liczba w nawiasach) oraz ile neuronów znajduje się w danej chwili w rozważanej ćwiartce (liczba po przecinku). Na przykład zapis:

*10 (3), 1*

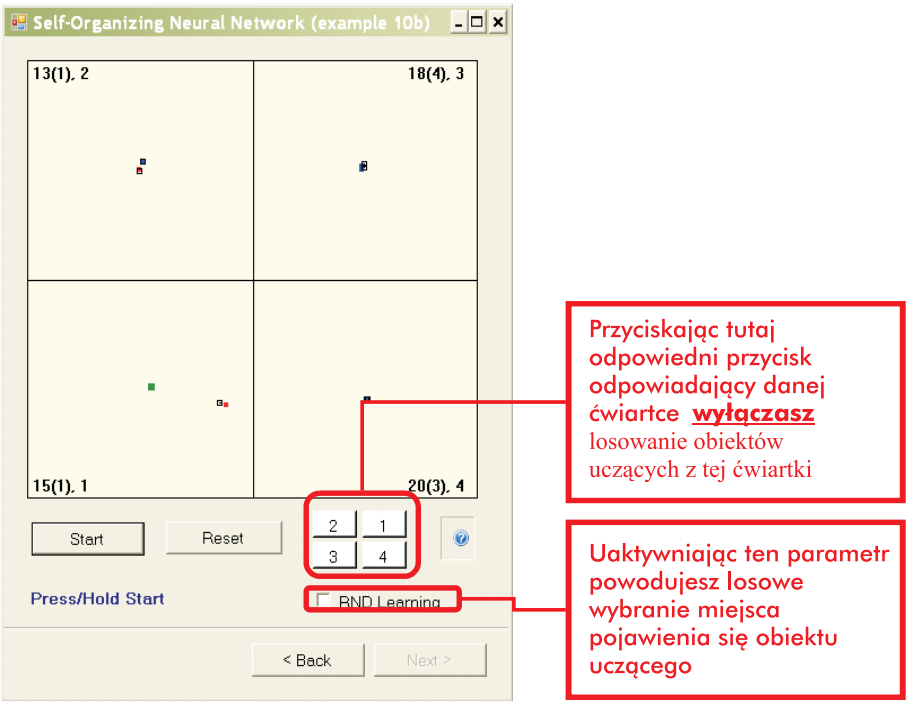
w danej ćwiartce oznacza, że obiekt uczący pojawił się w niej (do tej chwili) **dziesięciokrotnie** (z tego **trzykrotnie** wśród 9 pierwszych pokazów) i obecnie znajduje się w niej **jeden** neuron. Taka informacja jest szczególnie przydatna, gdy neuronów jest dużo i trudno z rzutu oka oszacować, ile z nich jest w jakiej okolicy, a także wtedy, gdy tworzące się w trakcie uczenia skupisko neuronów w pobliżu punktów powtarzalnego występowania obiektów ciągu uczącego – powoduje ich wzajemne przesłanianie się i utrudnia ocenę sytuacji.

Obserwowanie skutków pojawiających się (losowo) asymetrii, polegających na częstszym prezentowaniu obiektów z pewnej ustalonej ćwiartki (asymetrii ogólnej, a także dotyczącej obiektów pokazywanych na początku uczenia) – daje bardzo ciekawy materiał do przemyśleń.

Potem możesz zobaczyć, jak zachowują się większe sieci, złożone z kilkuset (maksymalnie do 1000) neuronów. **Naprawdę** warto to zobaczyć, jak równocześnie uczy się – na przykład – 700 neuronów! (Rys. 9.15 i 9.16). Program sam generuje początkowy rozkład wag dla wszystkich neuronów, a potem sam pokazuje wzorce sygnałów wejściowych (jako punkty położone w centrum odpowiednich – losowo wybieranych – ćwiartek). Po każdym pokazie następuje prezentacja „ruchu" wektorów wag pod wpływem procesu samouczenia, a następnie pokazywany jest następny obiekt wejściowy

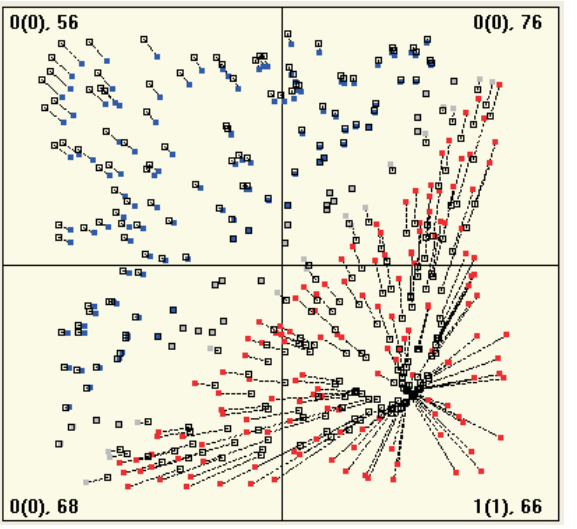



Rys. 9.13. Początek samouczenia sieci złożonej z 10 neuronów



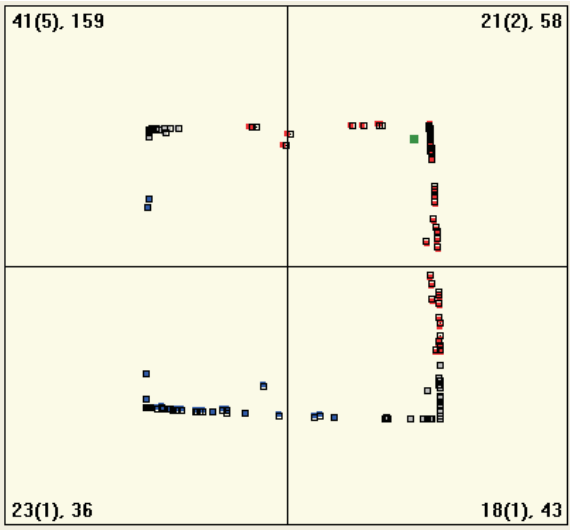

Rys. 9.14. Końcowy etap samouczenia sieci złożonej z 10 neuronów

i następuje kolejny krok samouczenia. Na początku, jeśli klikasz i zwalniasz przycisk **Start**, proces samouczenia przebiega, jak już wspomnieliśmy krokowo, to znaczy zatrzymuje się po każdym pokazaniu kolejnego obiektu, żebyś mógł się wygodnie przyjrzeć, co się w sieci dzieje. Potem te zatrzymania są automatycznie eliminowane i proces samouczenia rusza „pełną parą". Aby nie rozpraszać Twojej uwagi (zwłaszcza wtedy, gdy będziemy oglądać bardziej skomplikowane przykłady samouczenia), w dalszej części tego rozdziału rysunki będą przedstawiały tylko najważniejszą część okna (zawierającą rysunek rozmieszczenia punktów), w którym przeprowadzasz eksperymenty.



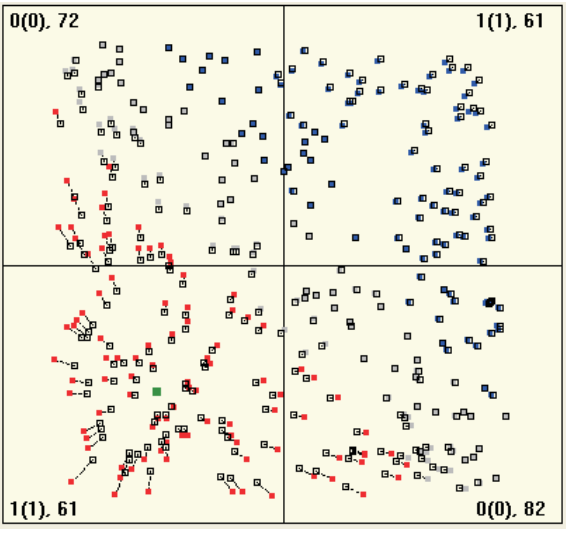

Rys. 9.15. Początkowy etap samouczenia sieci złożonej z 300 neuronów

Proces samouczenia, podobnie jak proces uczenia z nauczycielem, może być prowadzony przy różnych wartościach współczynnika uczenia. Na rysunku 9.15 pokazałem Ci, jak się zachowuje sieć, w której współczynnik uczenia jest bardzo duży. Taki „osobnik" jest ogromnie entuzjastycznie nastawiony do całego świata: każda nowa idea, każda nowa pojawiająca się koncepcja zaprząta jego uwagę bez reszty. Powoduje to, że pojawiający się atraktor przyciąga do siebie ogromną liczbę neuronów, które gorliwie i gwałtownie starają się jak najszybciej uzyskać zdolność do jego dokładnego rozpoznawania i lokalizowania tego właśnie nowego obiektu. Takie działanie sieci umożliwia wprawdzie jej szybkie uczenie, ale powoduje bardzo szybkie „nasycanie" jej możliwości poznawczych. Wystarczy niewielka (w stosunku do liczby neuronów sieci) liczba atraktorów, by wszystkie neurony zostały zaangażowane bez reszty w ich rozpoznawanie i w przypadku pojawiania się nowych wzorców nie ma już żadnych możliwości adaptacyjnych – sieć nowych idei




9.16. Końcowy etap samouczenia sieci złożonej z 300 neuronów

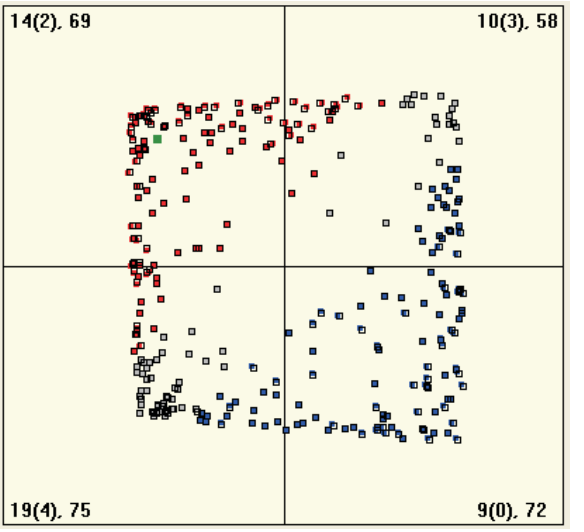
nie jest w stanie przyswoić ani zaakceptować. Odmiennie zachowuje się sieć przy małym współczynniku wzmocnienia. Tu na początku, po pojawieniu się nowego obiektu do rozpoznawania reakcje sieci są spokojne i wyważone (rys. 9.17). Na skutek tego postęp procesu uczenia nie jest nadmiernie szybki, co można by było uznać za wadę (por. rys. 9.18, na którym pokazano podobny stopień zaawansowania procesu samouczenia takiej „spokojniejszej" sieci, jak na rysunku 9.16 dla sieci „entuzjastycznej"). Jednak dzięki temu sieć stale




Rys. 9.17. Powolny i rozważny początek procesu samouczenia w systemie z niewielką wartością współczynnika uczenia ***Etha***

dysponuje pewnym zapasem neuronów, które jeszcze nie wyspecjalizowały się tak do końca, by nie móc przyswoić sobie i zaakceptować nowych wzorców sygnałów, czyli nowych idei, nowych wrażeń, nowych pojęć. I to także jest pewna wartość!

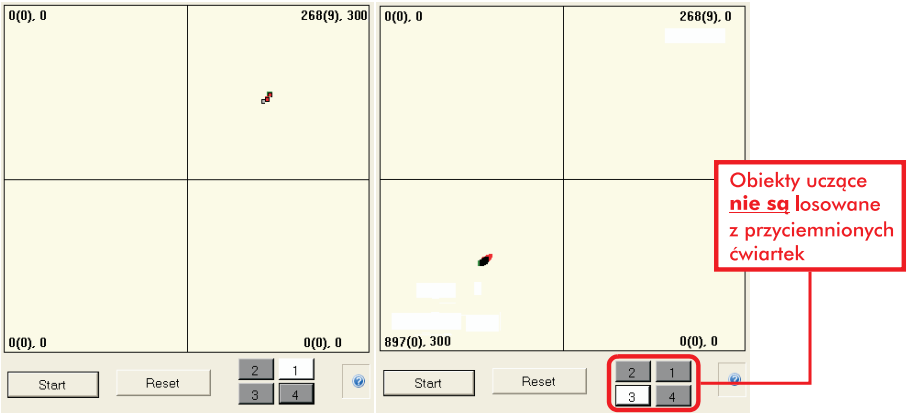
Zauważ, że podobnie bywa z ludźmi. Bywają tacy, którzy pierwszą napotkaną ideę (na przykład polityczną) przyjmują bezkrytycznie i entuzjastycznie, szybko opanowują jej arkana, identyfikują się z nią i emocjonalnie przeżywają. Tacy fanatycy nie są zdolni potem przyjąć jakichkolwiek nowych idei czy jakichkolwiek nowych poglądów – w ich mózgu po prostu nie ma już na nie miejsca. Być może jest to wyjaśnienie fenomenu instytucjonalnych morderców oraz terrorystów, którzy po szybkim przyswojeniu sobie jakiejś dostatecznie demagogicznej ideologii (islam w fanatycznej postaci, faszyzm, stalinizm) stają się nią tak dalece owładnięci, że w imię tych swoich (albo narzuconych im) urojeń są zdolni mordować i torturować ludzi, a takie naturalne i narzucające się idee, jak współczucie, humanitaryzm czy wreszcie zwykła ludzka przyzwoitość – nie mają do nich dostępu.




Rys. 9.18. Podobny etap procesu samouczenia, jak na rysunku 9.16, ale tym razem dla sieci z małą wartością współczynnika uczenia. Widać liczne „nie zagospodarowane" neurony

Prowadząc badania z taką silnie pobudliwą siecią neuronową możesz zaobserwować niekiedy ciekawe zjawisko: otóż jeśli taka „fanatyczna" sieć nie otrzymuje dostatecznie długo sygnałów ze strony atraktora, do którego została wcześniej przywiązana – dochodzi w niej często do gwałtownej „zmiany poglądów". Zaobserwujesz to w ten sposób, że nagle cała zbiorowość punktów, skojarzonych poprzednio z jednym wzorcem (czyli z jakąś jedną, całkowicie zaprzątającą umysł ideą) – nagle zaczyna migrować w kierunku zupełnie

innego wzorca, utożsamiając się z nim wkrótce równie skutecznie i równie krańcowo jak poprzednio z wcześniej przyswojonym wzorcem pierwotnym (rys. 9.19). To także nic zaskakującego, jeśli przyjrzysz się zachowaniom niektórych ludzi. Literatura pełna jest informacji o takich „nawróconych fanatykach", którzy jednak nie stają się spokojnymi obywatelami, lecz popadają z jednej skrajności w drugą. Niestety, również najnowsza historia dostarcza licznych przykładów oprawców, którzy najpierw zajadle torturowali przeciwników jakiegoś reżimu, a po jego obaleniu z równym zapałem przystępują do zwalczania wrogów nowego porządku.



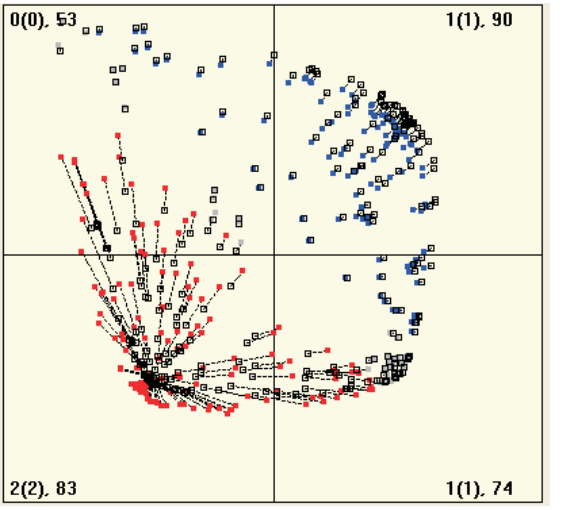

Rys. 9.19. Fascynacja wszystkich neuronów obiektami jednej tylko klasy (po lewej) oraz porzucenie tego jednego atraktora i przeniesienie wszystkich neuronów do rozpoznawania drugiego (po prawej). Zjawiska takie występują w sieci z dużą wartością współczynnika uczenia. Przemieszczenia widoczne na obu rysunkach wynikają z faktu, że obiekty ciągu uczącego należące do tej samej klasy nie są identyczne i sieć bez przerwy się doucza

Natomiast model sieci z niewielkim współczynnikiem uczenia (rys. 9.17) znamionuje człowieka, który nie zapala się, nie gorączkuje, u którego każda nowa idea musi odpowiednio dojrzeć, wykrystalizować się, okrzepnąć – zanim zostanie rzetelnie i szczegółowo przyswojona. Ludzie tacy są często odbierani przez otoczenie jako niezbyt błyskotliwi, co czasem (niesłusznie!) uważa się jako tożsame z określeniem – niezbyt inteligentni. Jeśli jednak taki człowiek ustali swoje preferencje – na ogół nie zmienia ich zbyt szybko ani zbyt łatwo, często pozostając wiernym swoim ideałom nawet podczas brutalnej i długotrwałej indoktrynacji.

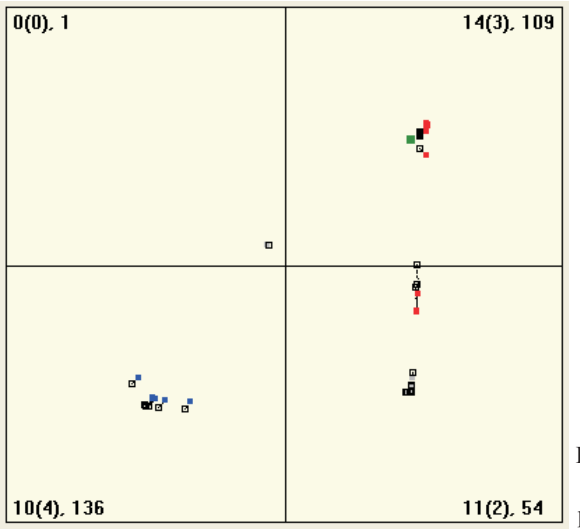
## 9.3. Czy postęp samouczenia można uznać za rosnącą mądrość sieci?

Podam Ci teraz pewne ogólne wskazówki, co mógłbyś zbadać przy pomocy opisanego wyżej programu, oraz zasugeruję Ci pewne proste, ale pouczające eksperymenty, które możesz sam wykonać. Myślę, że te eksperymenty zainteresują Cię zarówno z tego punktu widzenia, że pozwolą Ci lepiej poznać właściwości procesu samouczenia w sieciach neuronowych, jak i z tego powodu, że na ich podstawie można wyciągać bardzo ciekawe wnioski ogólniejszej natury – na przykład na temat tego, jak Ty sam zdobywasz i odkrywasz wiedzę. Opiszę Ci także przy tym, jak korzystać z programu **Example 10b**, chociaż jestem pewien, że po rozpoczęciu badań sam odkryjesz mnóstwo jego dalszych dodatkowych możliwości.

Uruchomiwszy proces uczenia z pewną liczbą neuronów, zaobserwuj najpierw proces wykrywania przez te neurony faktu, że mają do czynienia z powtarzającymi się prezentacjami pewnych charakterystycznych obiektów wśród danych wejściowych. Istotnie, bardzo szybko w chaotycznym z początku zbiorze punktów, reprezentujących położenia wag neuronów, zauważysz dążenie do podziału całej populacji neuronów na tyle podgrup, ile klas obiektów pojawia się na wejściu. Możesz to bardzo łatwo sprawdzić, ponieważ klikając w trakcie symulacji na obrazek z numerem określonej ćwiartki (od **1** do **4**) spowodujesz, że obiekty pokazywane w trakcie uczenia pochodzić będą tylko z trzech (albo z dwóch, albo nawet tylko z jednej), a nie z czterech ćwiartek. Obiekty z ćwiartki, którą nacisnąłeś, będą bowiem pomijane i będzie ona zaznaczona ciemniejszym kolorem. Ta opcja programu pozwoli Ci




Rys. 9.20. Początkowy etap procesu samouczenia, w czasie którego pomijane są obiekty z 2. ćwiartki




Rys. 9.21. Końcowy etap procesu samouczenia, w czasie którego pomijane są obiekty z 2. ćwiartki

samemu stwierdzić, jak skutecznie sieć sama z siebie wykrywa, ile (i jakich) klas obiektów powinna rozpoznawać (patrz rys. 9.20 i 9.21).

Zauważ, że w początkowym okresie nauki sieć „zagospodarowuje" częściowo te neurony, które potencjalnie mogły rozpoznawać obiekty brakującej klasy (rys. 9.20), w wyniku czego dochodzi do coraz większej „koncentracji uwagi" sieci na tych obiektach, które rzeczywiście w świecie występują i powinny być rozpoznawane. Jednak nawet w końcowym etapie procesu uczenia, kiedy sieć uzyskuje już doskonałą specjalizację w rozpoznawaniu i identyfikowaniu tych obiektów, które podczas samouczenia regularnie się pojawiały (trzy gęste skupiska na rysunku 9.21), w obszarze, w którym nie było żadnego atraktora, pozostaje pewna liczba „dyżurnych" neuronów. Są one rozsiane przypadkowo i w miarę dalszego kontynuowania procesu samouczenia będzie ich ubywać, ale jeszcze przez długi czas po zakończeniu podstawowej nauki w przypadku pojawienia się nowego obiektu w dotychczas „nie zagospodarowanym" obszarze – bardzo szybko znajdą się neurony, które podejmą proces nauki i wyspecjalizują się w jego identyfikacji. Dopiero po bardzo długim okresie systematycznego samouczenia, gdy sieć zgromadzi już bardzo duże „doświadczenie życiowe" – ten zasób wolnych neuronów wyczerpie się, gdyż wszystkie zostaną prędzej czy później „wciągnięte w orbitę wpływów" rzeczywiście występujących atraktorów. Jeśli po takiej petryfikacji sieci pojawi się zupełnie nowy obiekt – nie będzie już w sieci wolnych neuronów, które byłyby zdolne go zaakceptować i nauczyć się go identyfikować. Oznacza to, że długotrwały proces samouczenia prowadzi do utraty zdolności akceptowania i adaptowania kolejnych nowości – co zresztą znamy z codziennego do-

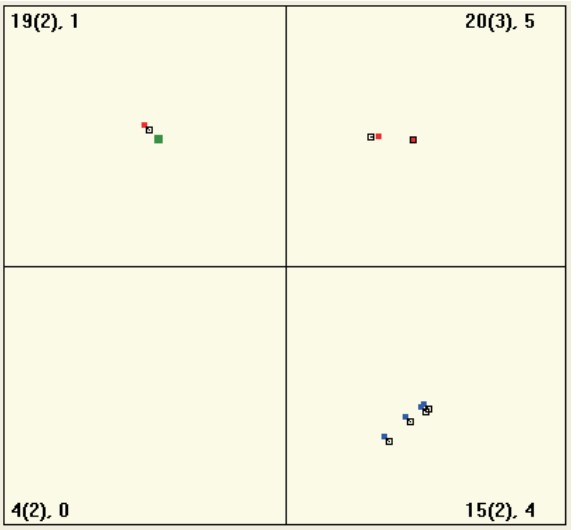
świadczenia: młodzi ludzie wykazują ogromną łatwość przy adaptowaniu się do nowych warunków i przy uczeniu się nowych umiejętności (na przykład korzystania z Internetu), starsi natomiast, doświadczeni i wyspecjalizowani w wykonywaniu pewnych funkcji i w identyfikowaniu pewnych sytuacji – uczą się z trudem, źle przyjmują nowości i mają skłonność do reagowania dezaprobatą na nowe zjawiska i procesy. Jak widzisz – sieć neuronowa robi prawie dokładnie to samo.

Szczegółowy przebieg omówionych wyżej procesów zależy od tego, jaki jest początkowy rozkład wartości współczynników wag neuronów (czyli od tego, jakie „osobnik" ma wrodzone cechy) oraz od tego, w jakiej kolejności pokazywane są poszczególne obiekty ciągu uczącego (czyli od tego, jakie „osobnik" ma doświadczenie życiowe). Jedni dłużej zachowują młodzieńczą zdolność do dziwienia się nowościom i entuzjastycznego ich przyjmowania, inni bardzo szybko zastygają w swoich starczych uprzedzeniach i fobiach. Możesz to także dokładnie prześledzić, ponieważ program **Example 10b** jest tak zbudowany, że naciśnięcie przycisku **Reset** powoduje ponowne wystartowanie procesu samouczenia (z taką samą liczbą neuronów) ponownie od losowo wybranych wartości początkowych wektorów wag wszystkich neuronów i z nowym (przypadkowym i innym niż poprzednio) ciągiem obiektów uczących, pochodzących z ćwiartek które są aktualnie wybrane. Klikając na poszczególnych przyciskach oznaczających poszczególne ćwiartki, możesz dowolnie włączać i wyłączać prezentacje obiektów z poszczególnych klas (także w trakcie procesu uczenia), więc de facto masz cały eksperyment stale pod kontrolą.

Ponawiając kilka razy proces uczenia – zauważysz łatwo, że dążenie do samoorganizacji (w sensie spontanicznego dążenia poszczególnych neuronów do identyfikowania się z określoną klasą nadchodzących sygnałów wejściowych) jest stałą cechą samouczącej się sieci neuronowej – niezależną od tego, jak wiele jest klas, jak liczna jest populacja neuronów i jak są rozłożone ich początkowe wartości współczynników wag. Zauważysz też, że sieci mające większą liczbę neuronów („bardziej utalentowane") wykazują dłużej zdolność do akceptowania nowości i uczenia się nowych umiejętności. Kostnienie w starczych uprzedzeniach i przesądach nie jest więc – jak się często twierdzi – wyrazem mądrości przychodzącej z wiekiem, lecz stanowi bezpośrednią manifestację przyrodzonej głupoty.

## 9.4. Co jeszcze warto zauważyć podczas samouczenia sieci?

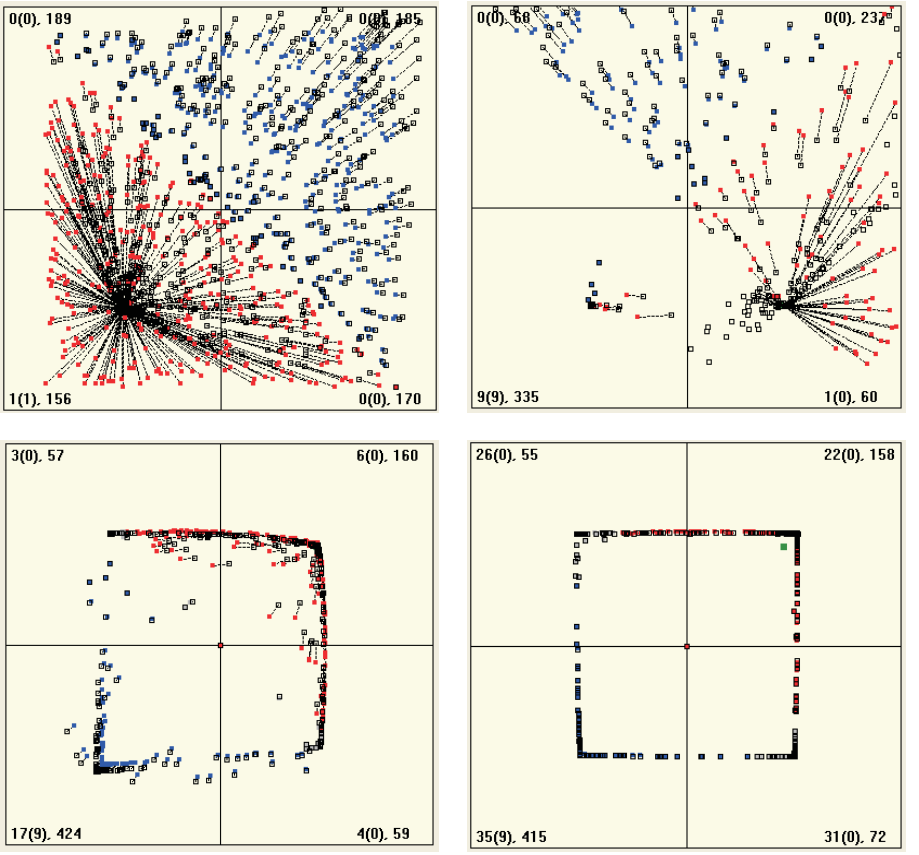
Przy użyciu programu **Example 10b** możesz wykonać bardzo wiele różnych badań. Na wstępie proponuję Ci, abyś przeanalizował **wpływ sposobu prezentacji obiektów ciągu uczącego na proces samouczenia sieci.**




Rys. 9.22. Nierówność reprezentacji klas powstała w następstwie procesu samouczenia

Tu już nie ma gotowych mechanizmów wbudowanych w program, które by Ci podawały odpowiednie fakty „na talerzu", ale obserwując wielokrotnie symulowany proces samouczenia dostrzeżesz z pewnością, że klasy, których obiekty pokazywane są częściej – przyciągają do siebie znacznie więcej neuronów niż te klasy, których obiekty pojawiają się rzadziej (por. na rysunku 9.22 wyniki dla 1. i 2. ćwiartki). Może to czasem prowadzić wręcz do tego, że dla pewnych pokazywanych klas może całkiem zabraknąć „chętnych" do ich rozpoznawania – zwłaszcza w sieciach o niewielkiej liczbie neuronów (por. 3. ćwiartkę na rysunku 9.22). Jest to spory problem, który sygnalizowałem Ci już w jednym ze wstępnych rozdziałów. Na skutek takiego właśnie zachowania się – sieci samouczące muszą zawierać o wiele więcej neuronów niż realizujące to samo zadanie rozpoznawania i klasyfikacji sieci uczone z nauczycielem.

Badając proces samouczenia na wielu przykładach zauważysz także, że szczególnie istotne znaczenie mają **pierwsze „doświadczenia" sieci.** Daje się to zauważyć na rysunku 9.23, na którym przytoczono obraz początku (po lewej u góry) i końca (po prawej u dołu) procesu samouczenia, w którym doszło do przypadkowego wylosowania w 9 pierwszych krokach algorytmu




Rys. 9.23. Ilustracja przebiegu samouczenia z początkową preferencją dla 3. klasy

wyłącznie obiektów należących do 3. ćwiartki. Zauważ, jak silnie wpłynęło to na końcowy rozkład liczby neuronów w poszczególnych ćwiartkach!

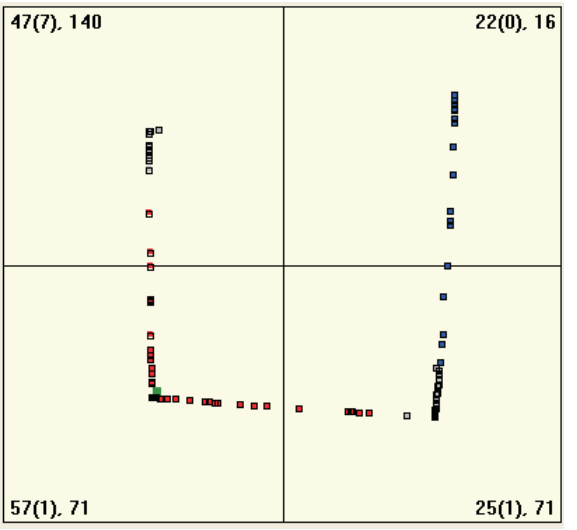
Zauważ, że ten przemożny wpływ „pierwszych doświadczeń" na kształtowanie się późniejszych kategorii pojęciowych i przyzwyczajeń można także zaobserwować u ludzi. Jeśli pewne ćwiartki są początkowo wyłączone („zubożenie skali doświadczeń w dzieciństwie"), a potem zostają włączone – to i tak reprezentacja tych pierwotnie dyskryminowanych na początku samouczenia wrażeń lub pojęć (reprezentowanych w naszym modelu przez ćwiartki, w których pojawiają się sygnały) pozostaje widoczna w sieci w postaci ograniczonej lub wręcz brakującej reprezentacji tych właśnie pojęć albo braku wrażliwości na określone sygnały. Obserwacja ta wyjaśnia, dlaczego tak bardzo ograniczona jest możliwość skorygowania skutków „trudnego dzieciństwa" w wieku dojrzałym.

Obserwując procesy „przyciągania" jednych neuronów i „odpychania" innych przez kolejne pokazy – możesz próbować wyobrazić sobie, co dzieje się w Twoim własnym mózgu, gdy obserwujesz nowe sytuacje i gdy Twój umysł usiłuje je analizować i specyfikować. W szczególności interesujące analogie do procesów zachodzących w rzeczywistej sieci neuronowej zauważysz, gdy uruchomisz symulację z dużą liczbą neuronów (popatrz na rysunek 9.23, który przedstawia samouczenie sieci zawierającej 700 neuronów). Masowe przemieszczanie się „stanów świadomości" wielu neuronów, zbiegających się w kierunku punktów odpowiadających zaobserwowanym bodźcom wejściowym, tak aby w następstwie tego stać się detektorami tych właśnie klas wejściowych sygnałów – jest czymś, co bez wątpienia trzeba samemu zobaczyć!

Jeśli chcesz, to możesz także sam zobaczyć, jakie „rewolucje" dzieją się w zbiorze neuronów podczas pierwszych kroków procesu samouczenia. Najlepiej zrobić to, przeprowadzając eksperyment krok po kroku. Informację na temat tego, które obiekty były częściej pokazywane w początkowym okresie uczenia, a które rzadziej – masz dodatkowo pokazaną w postaci liczb w odpowiednich ćwiartkach, możesz więc bez trudu zauważyć i przeanalizować podstawowe związki, jakie istnieją pomiędzy częstością prezentacji określonego obiektu w ciągu uczącym a liczbą neuronów, które będą gotowe go rozpoznawać. Omówiony efekt zaobserwujesz szczególnie wyraźnie, gdy porównasz wyniki wielu symulacji, ponieważ nie możesz zapominać o tym, że na wyniki każdej konkretnej **pojedynczej** symulacji silny wpływ ma także początkowy rozkład współczynników wagowych poszczególnych neuronów oraz **kolejność** prezentacji obiektów ciągu uczącego, a te oba czynniki są w moim programie dyktowane przez mechanizm przypadkowego losowania.

## 9.5. „Marzenia" i „fantazje" powstające podczas samouczenia sieci

Jak już zaobserwujesz proces tworzenia się i doskonalenia (precyzowania) spontanicznie odkrywanych przez sieć „pojęć" – możesz z kolei zająć się bardziej subtelnymi zjawiskami. Zauważysz zapewne (szczególnie „ćwicząc" przykłady z dużą liczbą neuronów), że poza głównym procesem tworzenia się **skupisk** neuronów rozpoznających główne podawane do systemu obiekty – pojawiają się także całe „ścieżki" neuronów, które spontanicznie dążą do wykrywania i rozpoznawania obiektów wejściowych o właściwościach **pośrednich** pomiędzy obiektami w rzeczywistości pokazywanymi (rys. 9.24). Dalszy proces samouczenia powoduje wprawdzie, że te „detektory hybryd"



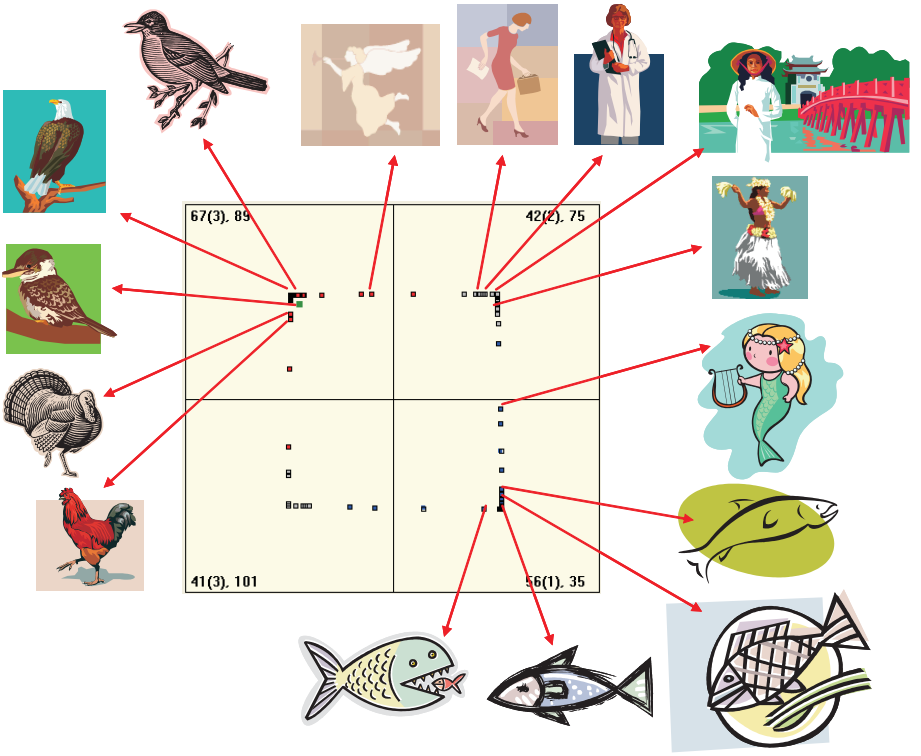

Rys. 9.24. Powstawanie neuronów (hybryd), które gotowe są wykrywać obiekty o własnościach pośrednich pomiędzy rzeczywiście występującymi

zostają wchłonięte przez „prawdziwe" ośrodki sygnalizujące realne obiekty – ale ten etap spontanicznego **fantazjowania** na temat bytów możliwych, chociaż nie istniejących – powtarza się z zadziwiającą regularnością.

Pomyśl, co to w istocie znaczy? Otóż sieć, której pokazano wielokrotnie rybę i kobietę, będzie potrafiła rozpoznać zarówno kobietę, jak i rybę (w odpowiednich punktach przestrzeni sygnałów wejściowych będą ulokowane neurony, które będą odpowiednio rozpoznawały pojawiające się na wejściu wizerunki kobiet i wizerunki ryb). Ale w sieci pojawiły się także neurony, które „rozpoznają" (a raczej **są gotowe** rozpoznawać) twory, których cechy będą mieszaniną cech ryby (ogon) i cech kobiety (głowa i ramiona). Takich tworów podczas uczenia nie pokazywano – a jednak sieć neuronowa przygotowała neurony do ich wykrywania, czyli w pewnym sensie sama je sobie „wyobraziła".

Na podobnej zasadzie ta sama sieć, która w innej grupie neuronów wytworzyła sobie zdolność rozpoznawania różnych ptaków, potrafi skojarzyć cechy ptaka (skrzydła) z cechami kobiety (głowa, ręce, suknia) tworząc wyobrażenie … anioła (patrz rys. 9.25).

Powtarzając różne doświadczenia, możesz się przekonać, że każda „młoda" (nie do końca jeszcze nauczona) sieć neuronowa będzie miała tendencję do „wymyślania" takich różnych nie istniejących bytów. Jeśli kolejne doświadczenia będą uporczywie potwierdzały, że **istnieją** kobiety i **istnieją** ryby, ale **nie istnieją** twory mające cechy zarówno kobiety, jak i ryby, czyli bajkowe ***syreny*** – odpowiednie neurony przekwalifikują się (chociaż zauważ podczas

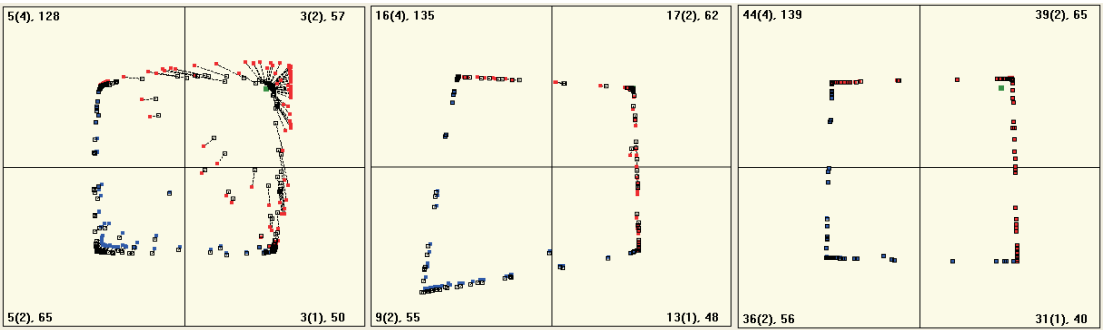



Rys. 9.25. Reprezentacje rzeczywistych oraz wyimaginowanych obiektów w samouczącej się sieci neuronowej

prowadzonych eksperymentów, jak niechętnie to robią!) na rozpoznawanie tworów rzeczywistych, a nie wykombinowanych fantasmagorii. Jeśli jednak taki hybrydowy twór istnieje (mysz ze skrzydłami = nietoperz) – to wcześniejsze doświadczenia ze „składowymi" obiektami zdecydowanie ułatwią jego wykrycie i sklasyfikowanie.

Zauważ, że ta zdolność do „wymyślania" tworów nie istniejących, a będących pewną kombinacją elementów pochodzących z odległych niekiedy od siebie wrażeń i doświadczeń – jest cechą sieci wyłącznie w początkowym stadium samouczenia, a więc sieci młodych. Może jest to jakieś wyjaśnienie znanej skłonności młodych dzieci do bajek i legend, które nudzą lub nawet drażnią ludzi dorosłych i bardziej doświadczonych? Może w tym także tkwi wyjaśnienie fenomenu rozpoetyzowanej fantazji bardzo młodych cywilizacji (starożytna Grecja) i tępego pragmatyzmu starych i skostniałych społeczeństw końca XX wieku? Może obserwując (rys. 9.26) powstające w trakcie samouczenia wątłe łańcuszki neuronów, próbujących długo, wbrew oczywistym

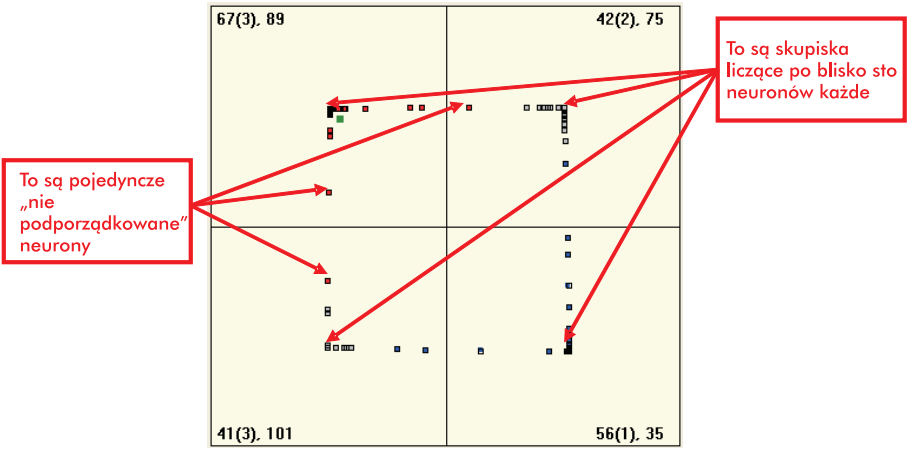
faktom, zachować **nie istniejącą** w rzeczywistości więź pomiędzy wykrywanymi i konkretnie lokalizowanymi zjawiskami – w istocie śledzisz neuronowe podstawy procesu tworzenia się w niektórych umysłach błyskotliwych skojarzeń, odległych analogii, zaskakujących przenośni? Może takie „mostki" spontanicznie powstających, ale potem rwących się jak pajęczyna pod naporem nowych faktów, produktów skojarzeń i fantasmagorycznych rojeń, fantastyczne kombinacje cech istniejących obiektów, kreowane przez same mechanizmy neuronowe w celu wytworzenia czegoś, co może i nie istnieje, ale jest piękne i ciekawe – są istotą poezji, sztuki, a także kreatywnej części nauki? I może docierasz do zrozumienia najgłębszych powodów znanego faktu, że poeci (i prawdziwi uczeni) są trochę jak dzieci...




Rys. 9.26. Samorzutne powstawanie i niszczenie skojarzeń w trakcie procesu samouczenia

Czasem podczas eksperymentów będziesz obserwował, że niektóre (zwykle pojedyncze) neurony pozostają uparcie poza obszarami skupionymi wokół miejsc, w których pojawiają się wejściowe obiekty. Tak naprawdę nie mają one praktycznego znaczenia, ponieważ zdecydowana większość neuronów grzecznie podlega uczeniu i potulnie poddaje się coraz doskonalszej specjalizacji, zmierzającej do idealnego wykonywania zadania, polegającego na wykrywaniu i sygnalizowaniu tych obiektów, które w rozważanym świecie naprawdę się regularnie pojawiają. A jednak te oderwane od rzeczywistości neurony pozostają i trwają (rys. 9.27) przy swoich wyobrażeniach skrzydlatych węży czy wielogłowych smoków – jako marzeniach wracających w snach...

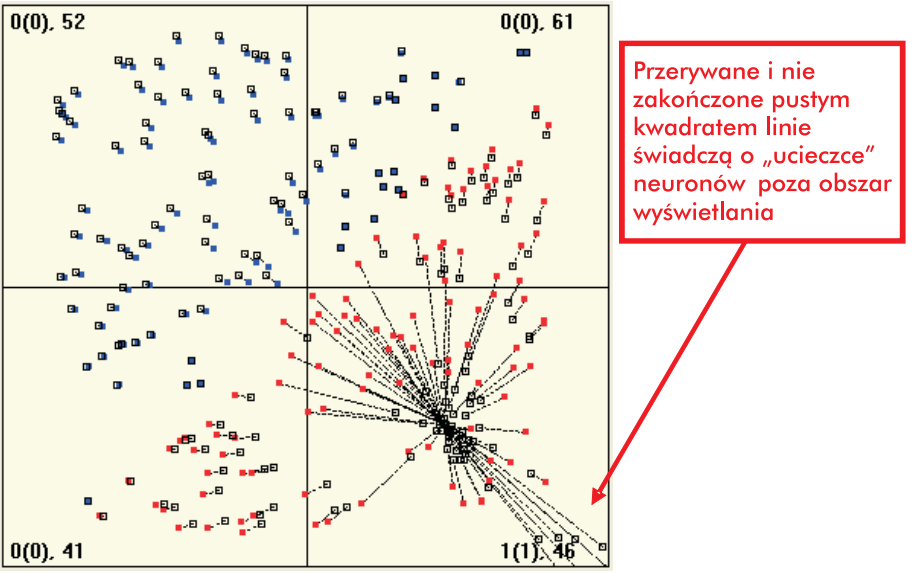
Wróćmy jednak do dalszych eksperymentów. Jeśli uważnie będziesz śledził początkowe fazy procesu samouczenia – wówczas w niektórych (raczej rzadkich!) przypadkach zauważysz jeszcze jedno ciekawe zjawisko. Otóż po pokazaniu obiektu w pewnym punkcie przestrzeni sygnałów wejściowych – niektóre neurony (bardzo nieliczne na ogół, ale tym bardziej godne uwagi) zmieniają swoje początkowe „przyzwoite" położenie na nowe położenie,



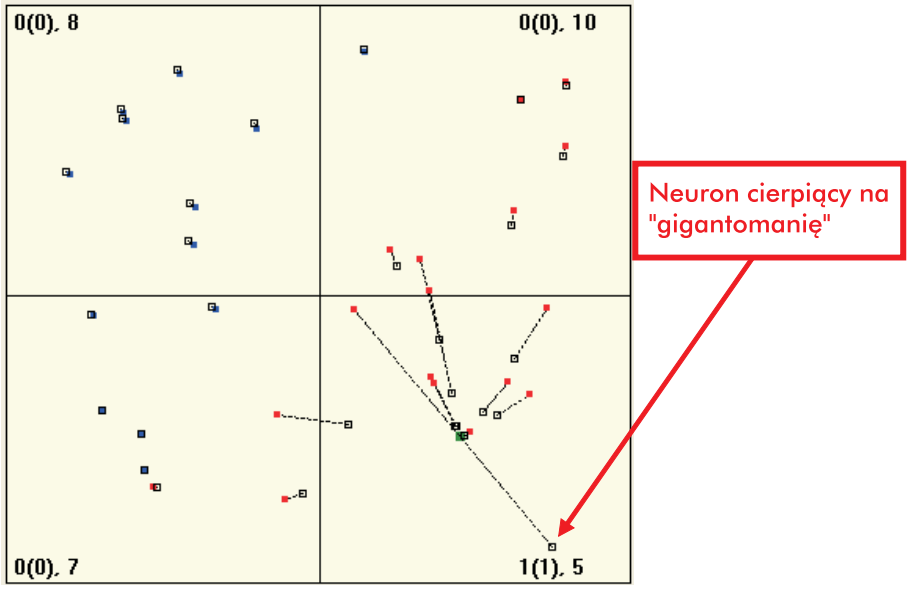

Rys. 9.27. Występowanie pojedynczych neuronów poza globalnymi skupiskami przy bardzo zaawansowanym procesie samouczenia (po wykonaniu blisko dwustu kroków)

bardzo odległe od początkowego, znajdujące się w tym samym kierunku, co pokazany obiekt wejściowy – ale znacznie dalej od początku układu współrzędnych. Jak powiedziałem – to zjawisko występuje raczej rzadko, bo zależy silnie od początkowego rozkładu wartości współczynników wag synaptycznych (w końcu nie każdy rodzi się poetą...), ale cierpliwie ponawiając doświadczenia, na pewno je zauważysz (rys. 9.28). Takie „zbuntowane" neurony z reguły lądują już poza granicami ramki ograniczającej rysunek i dlatego można je zaobserwować głównie po tym, że na rysunku pojawią się linie nie zakończone docelowym, pustym kwadratem, co sugeruje, że jest on poza obszarem wyświetlania.

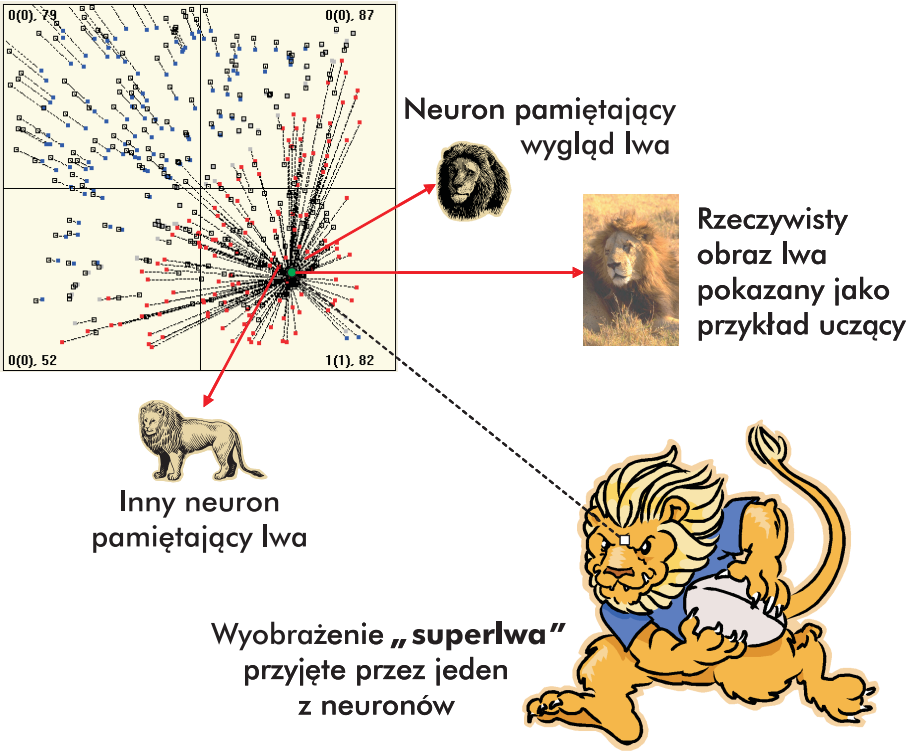
Neurony uciekające daleko na początku procesu samouczenia są potem, w kolejnych etapach procesu uczenia, nieuchronnie ściągane przez obszar związany z rzeczywistym położeniem prezentowanego wzorca i ostatecznie trafiają do tego obszaru. Można by się nimi wcale nie przejmować, gdyby nie fakt, że ilustrują one pewne znane z psychologii zjawisko – mianowicie **gigantomanię** (rys. 9.29). Istotnie, zauważ, że taki „wyskakujący" poza obszar wyświetlania neuron staje się wzorcem (wewnętrznym wyobrażeniem) obiektu, który ma z grubsza takie same cechy, jak rzeczywiście dostrzeżony na początku uczenia obiekt wzorcowy – ale wszystkie te cechy są znacznie **większe**. Taki sobie neuronowy Guliwer. Jeśli pokazanym obiektem jest lew – to wyobrażeniem powstałym w wyniku powiększenia jego realnie dostrzeżonych cech – jest monstrum mające o wiele większe rozmiary, potężniejsze kły i pazury, bardziej zmierzwioną grzywę, groźniejszy ryk, bardziej cuchnący oddech itp. – patrz rys. 9.30. Takie przesadne wyobrażenia nie trwają długo,




Rys. 9.28. Efekt dalekiej ucieczki neuronów w początkowym etapie procesu samouczenia




Rys. 9.29. Efekt „gigantomanii" na nieco mniejszą skalę, niż na rysunku 9.28, ale także zauważalnej

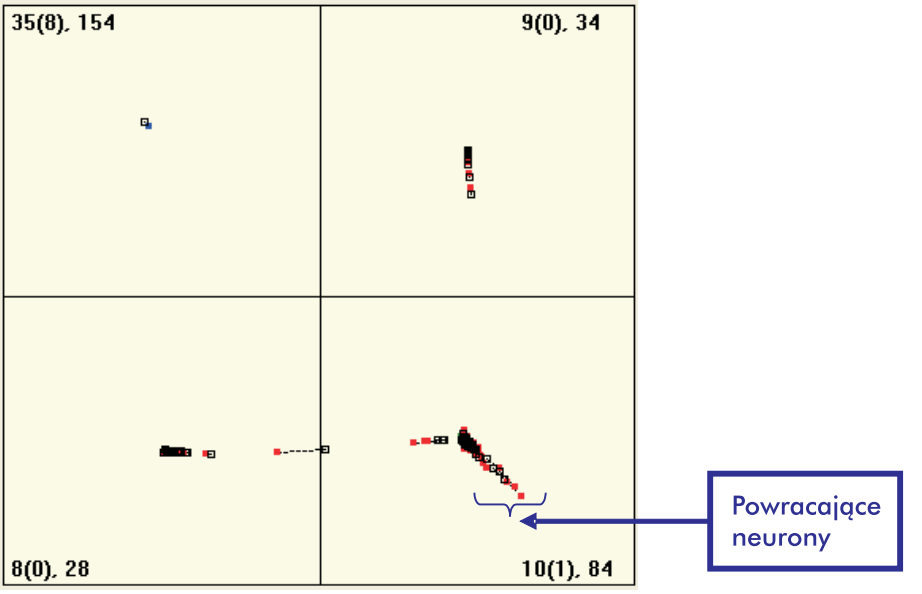



Rys. 9.30. Interpretacja zjawiska „gigantomanii"

bo weryfikuje je i ustawia we właściwej skali samo życie i niesione przez nie doświadczenia. Ale dla naiwnej i zapalczywej Młodości wkraczającej dopiero w świat nowych doznań – ta pierwsza Dziewczyna jest zawsze Naj...! Czy potrafisz zauważyć i docenić, że wymyślona przeze mnie i badana przez Ciebie sieć neuronowa robi dokładnie to samo?

Najbardziej smutne zjawisko możesz zaobserwować w tych doświadczeniach pod koniec procesu samouczenia. Większość neuronów tworzy już zwarte skupiska ciasno zebrane wokół miejsc, w których pojawiają się wejściowe obiekty (każdy już wybrał swego Pana i gorliwie mu służy...) – i wtedy, niekiedy nawet spoza obszaru wyświetlania, wyłaniają się takie „niedostosowane" osamotnione punkty, nieubłaganie ściągane w obręb obszarów rozpoznawania realnie występujących obiektów, ale wyraźnie stawiające opór (rys. 9.31). Ich poglądy są odmienne od poglądów większości, co gorsza – codzienność pokazuje, że są to poglądy błędne – one jednak wolą trzymać się pięknej bajki, niż uznać prawa realnego świata, gdzie liczą się głównie inne wartości ... No, ale zostawmy to i wróćmy do sieci neuronowej. Po pewnym czasie

„zbuntowane" neurony zostają wchłonięte przez masę, dostosowane do większości, sprowadzone na prawdziwą drogę. Jednak w tym ich buncie jest coś pięknego. Ponadto zauważ, że w przypadku, kiedy w świecie dostarczającym sieci neuronowej wejściowych „wrażeń" pojawi się coś nowego i naprawdę nieoczekiwanego – właśnie tacy nie dostosowani marzyciele mają szansę na wielką wygraną, której nie zdobędzie wtedy tłum idealnie dostosowanych konformistów. Szkoda tylko, że zdarza się to tak rzadko – nawet w sieciach neuronowych!



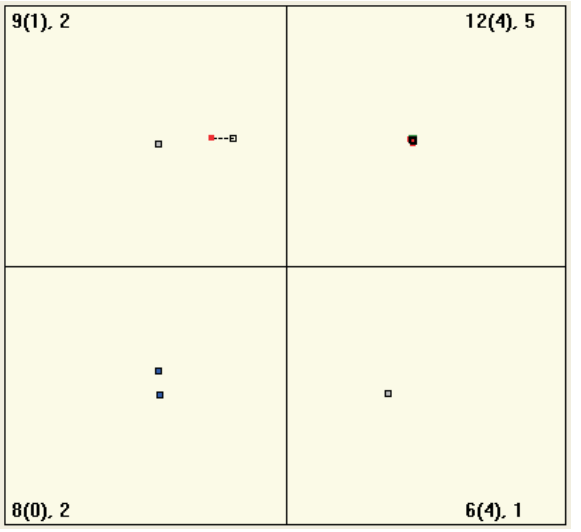

Rys. 9.31. Przyciągnięcie do realnego atraktora neuronów, które uciekły w początkowym etapie procesu uczenia, tworząc fikcyjne reprezentacje danych opisane wyżej pod nazwą „gigantomanii"

## 9.6. Zapamiętywanie i zapominanie

W całym tym rozdziale zajmowaliśmy się głównie procesem zapamiętywania informacji i zdobywania wiedzy w trakcie procesu samouczenia. Jednak z codziennego doświadczenia znasz na pewno inne zjawisko, także często występujące – mianowicie zjawisko **zapominania**. Jest ono uciążliwe (zwłaszcza gdy pojawia się przed ważnym egzaminem…), ale z biologicznego punktu widzenia jest potrzebne. Każde otoczenie, w którym żyje określony organizm (na przykład Ty…), podlega ustawicznym zmianom, zatem sposoby działania wyuczone i skuteczne na pewnym etapie życia – stają się nieaktualne (a cza-

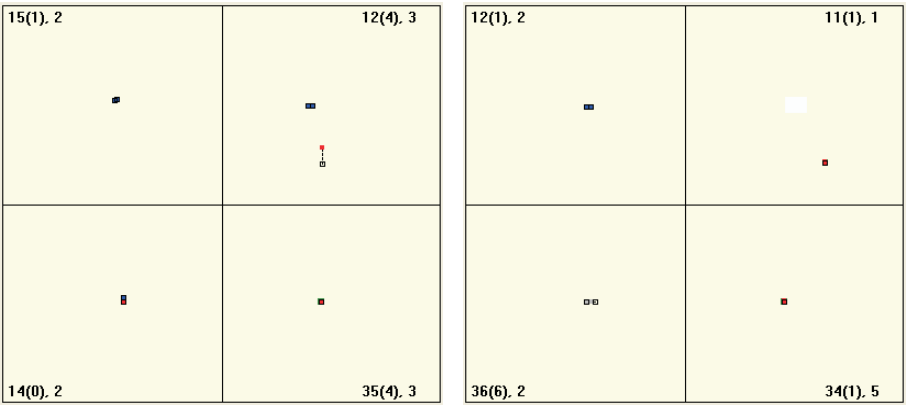
sem wręcz szkodliwe) w świetle nowych doświadczeń. Ten fakt (między innymi) zmusza do tego, że trzeba ustawicznie zdobywać nowe umiejętności – **zapominając** równocześnie te poprzednie, bo inaczej mogą się mylić albo mieszać.

Spróbuj zauważyć i przeanalizować to zjawisko w swojej symulowanej sieci neuronowej. Otóż podejmując badania z niewielkimi ilościami samouczących się neuronów, możesz łatwo zauważyć, że często pojawiające się **nowe** obiekty mogą „odciągać" neurony od innych, rzadziej pojawiających się klas, które mogłeś uważać za dobrze już wyuczone. W skrajnych przypadkach może dochodzić wręcz do swoistego „porywania" przez często powtarzające się **nowe** obiekty tych neuronów, które wcześniej już całkiem dobrze lokalizowały jakieś inne wzorce, wcześniej pracowicie wyuczone, które jednak obecnie przestały się pojawiać. Na rysunku 9.32 masz widok sytuacji, w któ-




Rys. 9.32. Początek procesu zapominania, występującego podczas samouczenia. Obecnie sieć posiada jeszcze „starą wiedzę" i najlepiej rozpoznawana jest klasa nr 1

rej sieć stosunkowo nieźle rozpoznaje wszystkie cztery pokazywane jej wzorce. Zauważ, że najsilniej reprezentowany jest wzorzec nr 1 (z 1. ćwiartki). W tym momencie proces samouczenia zostaje (celowo) zakłócony – wszystkie inne obiekty są pokazywane nadal, natomiast obiekty z klasy nr 1 – nie. Po bardzo krótkim czasie dochodzi w tej sytuacji do „rabunku". Neurony, które rozpoznawały obiekty klasy 1, pospiesznie przekwalifikowują się i zaczynają specjalizować się w rozpoznawaniu innej klasy (w pokazanym przykładzie – klasy nr 4 – patrz rys. 9.33 lewa strona). W ten sposób klasa, która była poprzednio szczególnie dobrze rozpoznawana – popada w zapomnienie. Nie całkiem, bo nawet po bardzo długim procesie uczenia pozostał pewien ślad




Rys. 9.33. Stopniowe zapominanie klasy nr 1 w sytuacji, kiedy jej ślad pamięciowy nie jest systematycznie wzmacniany w toku dalszego procesu samouczenia

pamięciowy, który sygnalizuje, że kiedyś taka klasa była (rys. 9.33 po prawej stronie), jednak ślad ten jest słaby (tylko jeden neuron) i mocno zniekształcony (położenie tego neuronu uległo znacznemu przemieszczeniu w stosunku do położenia pierwotnego).

Zauważ, że z Twoim własnym umysłem bywa podobnie – na przykład podczas wakacji możesz zainteresować się botaniką i poznać nazwy wielu roślin, a także nauczyć się je identyfikować i klasyfikować. Jeśli jednak nie będziesz do tej wiedzy wracał i stale ją utrwalał – nabyte pojęcia i kryteria rozpoznawania zostaną zatarte przez nowe informacje (na przykład umiejętność rozróżniania nowych marek samochodów), w wyniku czego już na początku kolejnego lata powiesz z westchnieniem: *Jaka piękna łąka! Ile kwiatów!* ***Kiedyś*** *wiedziałem, jak się każdy z nich nazywa...*

## 9.7. Czy każde dane wejściowe spowodują samouczenie się sieci?

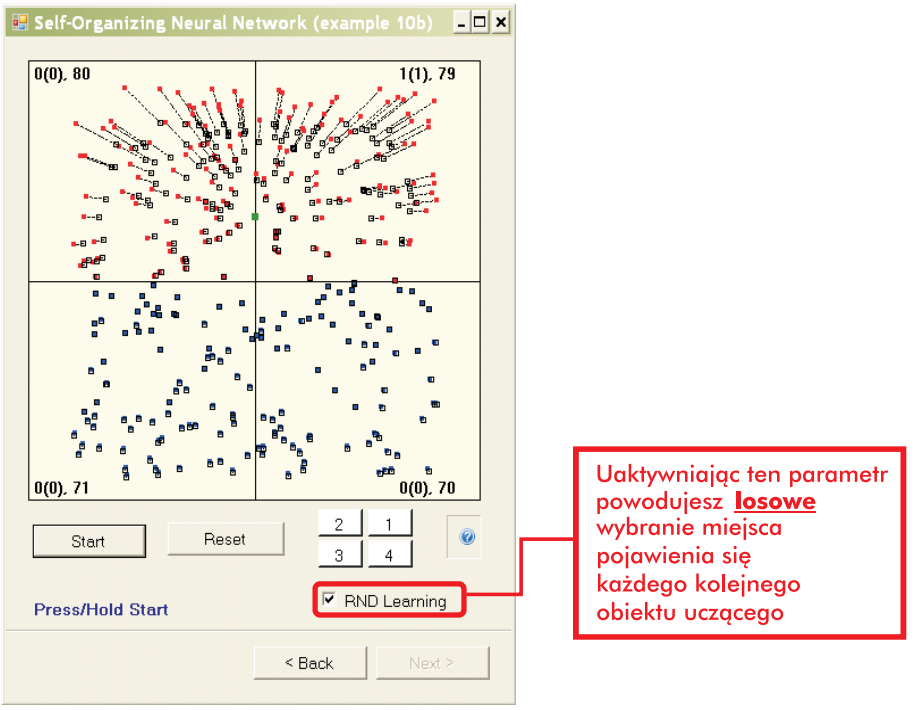
Pozostawmy jednak na uboczu smutny problem zapominania, omówiony w treści poprzedniego podrozdziału i powróćmy do badania, jak działa samoucząca się sieć neuronowa. Prowadząc badania z programem **Example 10b**, mogłeś stwierdzić i zaobserwować jedno niepokojące zjawisko: w rezultacie spontaniczności procesu samouczenia – reprezentacja w sieci neuronowej jednych klas może być bardziej liczna (wiele neuronów wykrywa i sygnalizuje obiekty tych właśnie klas), inne natomiast klasy obiektów mogą być reprezentowane słabo – lub zgoła wcale! Jest to duży mankament rozważanych tu metod samouczenia, którym zajmiemy się dokładniej w podrozdziale 9.8.

Teraz natomiast chcę jeszcze raz podkreślić i naocznie Ci pokazać, że proces samoorganizacji i samouczenia **tylko wtedy** zachodzi, jeśli w wejściowym ciągu danych uczących **istnieje** pewna ustalona **prawidłowość**, na której sieć się może oprzeć. W dotychczasowych eksperymentach z programem **Example 10b** miałeś sytuację grzeczną i dobrze ustawioną, ponieważ obiekty, które były „pokazywane" sieci – należały do pewnej liczby (zwykle czterech) ustalonych klas i zajmowały w przestrzeni sygnałów położenia dobrze wzajemnie odseparowane, mianowicie znajdujące się w centrum poszczególnych ćwiartek układu współrzędnych. Obiekty te pojawiały się wprawdzie losowo, ale **nie** były lokowane w całkiem losowych miejscach. Każdorazowo pokazywany sieci punkt lokowany był (z pewnym rozrzutem) w okolicy centrum odpowiedniej ćwiartki, a więc był to jakiś (niekoniecznie całkiem typowy, bo ten byłby dokładnie w środku ćwiartki) przykład Marsjanina, Marsjanki itd. Przy takiej prezentacji obiektów obserwowałeś dobrze Ci już znane zjawisko: samouczenie sieci powodowało pojawianie się skupisk neuronów, coraz dokładniej i precyzyjniej specjalizujących się w rozpoznawaniu tych właśnie pokazywanych „wzorców", czyli wypadkowych wielu pokazów obiektów z danej klasy, różniących się wprawdzie nieco od siebie, ale jednak będących odmianami tego właśnie idealnego wzorca.

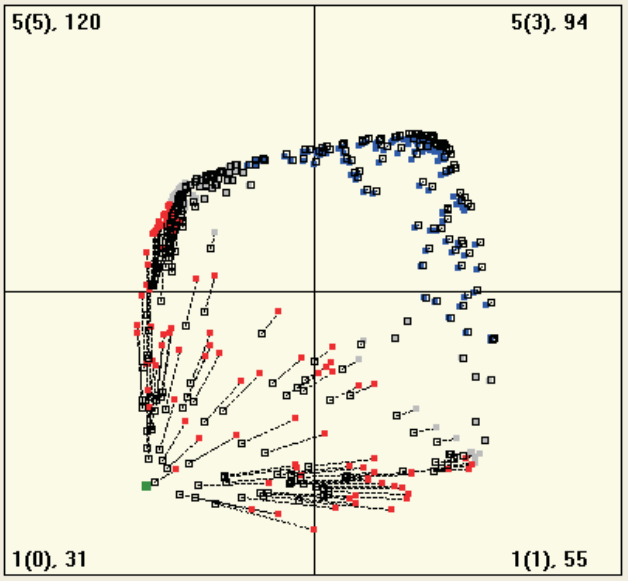
Zastanówmy się jednak teraz, jak się zachowa sieć, w której proces uczenia będzie skojarzony z **całkowicie losowym** wyborem położenia pokazywanych punktów. Wróćmy w tym celu do naszego przykładu z sondą kosmiczną wysłaną na Marsa. Powiedzmy, że Marsjanie nie istnieją i lądownik odbiera za pomocą swoich sensorów jedynie obrazy przypadkowych kształtów kręconych wichrem tumanów marsjańskiego pyłu. Możesz to łatwo prześledzić, ponieważ w programie **Example 10b** wbudowana jest także i taka możliwość. Wystarczy więc, że podczas symulacji uaktywnisz opcję **RND learning** (zaznaczysz myszką odpowiedni checkbox, który domyślnie do tej pory nie był uaktywniony). Od tej pory nie będziesz widział, w której ćwiartce pokazuje się obiekt uczący (rys. 9.34), gdyż będzie to mało ważne (pod warunkiem, że dopuścisz losowanie ze wszystkich ćwiartek – tak jak na rys. 9.34) – natomiast obiekt uczący będzie się ukazywał to tu, to tam, w obrębie całego układu współrzędnych, bez żadnego sensu i bez żadnego porządku.

Taki brak porządku spowoduje oczywiście to, że w wejściowym ciągu pokazów nie będzie już literalnie **żadnej** informacji. Neurony raz będą przyciągane do wnętrza obszaru, innym razem na zewnątrz (rys. 9.35).

Proces samouczenia nie doprowadzi w takim przypadku do wyodrębnienia wśród neuronów żadnych wyraźnych skupisk, natomiast zaczną się one grupować w formie dużego okręgu, gdyż tam właśnie lokować się będą lokalnie **uśrednione** wzorce sygnałów. Obserwując proces samouczenia w tym dziw-

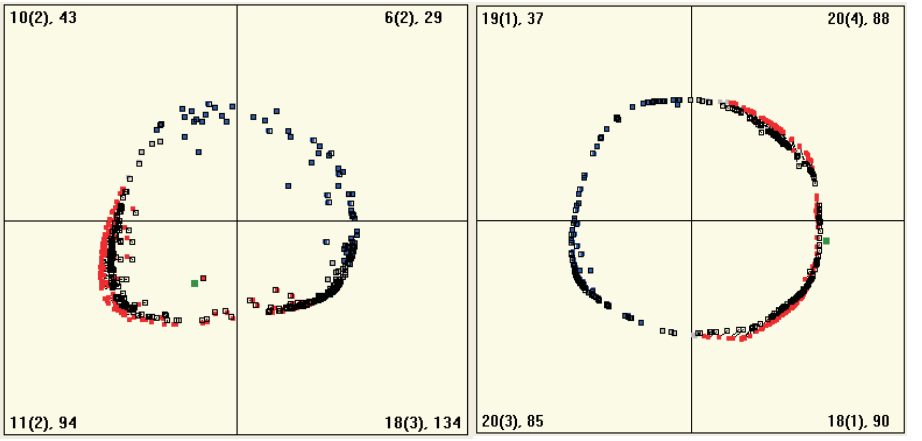



Rys. 9.34. Początek procesu samouczenia z obiektami wejściowymi pokazywanymi losowo




Rys. 9.35. Chaotyczne przemieszczenia neuronów przy losowo pokazywanych obiektach

nym przypadku dostatecznie długo – zauważysz, że okrąg ten nieznacznie modyfikuje swoje rozmiary i położenie w wyniku kolejnych pokazów (rys. 9.36), ale do wyodrębnienia jakichś wyraźnie wydzielonych skupień jednak nie dochodzi – no bo takich skupień nie ma także w danych wejściowych.




Rys. 9.36. Zaawansowany etap chaotycznego uczenia

Wniosek, jak i z tych badań wynika, jest częściowo optymistyczny, a częściowo pesymistyczny.

Optymistyczna część wniosku jest taka: Proces samouczenia sieci neuronowej może pomagać w wykryciu nieznanych prawidłowości, które są zawarte w danych dostarczanych na wejście sieci, przy czym użytkownik sieci nie musi wiedzieć, jakie są to prawidłowości ani ile ich jest. Niekiedy mówi się o tym, że samoucząca się sieć potrafi ***udzielać odpowiedzi na nie zadane pytania,*** a ujmując rzecz jeszcze bardziej ogólnie można powiedzieć, że sieci takie nie tylko gromadzą wiedzę (poprzez proces uczenia), ale także samodzielnie ***odkrywają wiedzę.*** Fakt ten bardzo nas cieszy, bo w informatyce mamy mnóstwo dobrych narzędzi, zdolnych do udzielania mądrych odpowiedzi na mądre pytania (na przykład Internet i systemy baz danych). Czasem udaje się uzyskać także mądrą odpowiedź na głupie pytanie – i to także bywa pożyteczne. Ale do tego, żeby odpowiadać na pytania, które wcale nie zostały postawione, mamy raczej niewiele narzędzi. Jak się przekonałeś, samoucząca się sieć neuronowa ma takie możliwości – i to jest jej bardzo miła cecha. Dodam, że czasem takie zadania szukania odpowiedzi na nie zadane pytania określa się angielskim terminem *Data Mining* i wykorzystuje się na przykład do wykrywania zachowań klientów w supermarkecie albo do ustalania preferencji abonentów telefonów komórkowych. Specjaliści od marke-

tingu potrafią zrobić użytek z każdej wiadomości o istnieniu powtarzalnych i masowych zachowań klientów, więc wiadomości takie są bardzo intensywnie poszukiwane.

Pesymistyczna część wniosku jest taka: Jeśli sieć jest poddawana procesowi samouczenia z użyciem danych nie niosących żadnej wartościowej informacji, na przykład danych w pełni losowych (a więc nie mających żadnych jawnych ani ukrytych znaczeń) – to nawet najdłuższe samouczenie sieci nie prowadzi do żadnych sensownych wyników.

Dlaczego ten wniosek uważam za pesymistyczny? Bo to oznacza, że w wymyślaniu rzeczy całkowicie nowych, takich, o których mówimy, że ich pomysł się wziął „z niczego", sieć nas jednak nie potrafi zastąpić!

A może to właśnie jest optymistyczna wiadomość?

## 9.8. Co może dać wprowadzenie konkurencji do sieci?

Samouczeniu można poddawać sieci dowolnego typu, ale szczególnie ciekawe wyniki otrzymuje się, gdy proces samouczenia sieci połączy się z jeszcze jednym procesem – konkurencji. **Konkurencja** (rywalizacja) między neuronami nie jest dla Ciebie tak całkiem nowym zjawiskiem. Już w podrozdziale 4.8 opisałem Ci (pamiętasz? jak nie, to zajrzyj do programu **Example 02b**) między innymi to, jak wyglądają i jak działają sieci, których neurony „konkurują" czy też „rywalizują" ze sobą. Jak zapewne pamiętasz, w sieciach z konkurencją wszystkie neurony otrzymują sygnały wejściowe (generalnie – te same, ponieważ sieci takie z reguły są jednowarstwowe), wszystkie obliczają sumy tych sygnałów (oczywiście przemnożonych przez odpowiednie – dla każdego neuronu inne – współczynniki wagowe), a następnie odbywa się „konkurs". Polega on na tym, że obliczone przez wszystkie neurony wartości wyjściowe są porównywane ze sobą i wyłaniany jest ***„zwycięzca"***, czyli ten neuron, który najsilniej zareagował na dany wejściowy sygnał.

Jak pamiętasz, pobudzenie neuronu jest więc tym silniejsze, im lepsza będzie zgodność między sygnałami wejściowymi podanymi w danej chwili a wewnętrznym wzorcem. Wiedząc, jakie neuron ma współczynniki wagowe (na podstawie ich położenia w przestrzeni sygnałów wejściowych), wiesz zatem, które neurony powinny zwyciężać w przypadku pokazywania próbek sygnałów wejściowych w poszczególnych miejscach przestrzeni sygnałów wejściowych. Łatwo będzie Ci to odgadnąć, ponieważ tylko ten neuron, którego wewnętrzna wiedza jest w największym stopniu zgodna z aktualnie podanymi sygnałami – wygra w konkurencji z innymi i tylko jego sygnał wyjściowy pojawi się na wyjściu sieci, a wszystkie pozostałe będą musiały milczeć. Oczywiście, taki „sukces" neuronu jest krótkotrwały, bo w następnej

chwili pojawi się inny zestaw sygnałów wejściowych i konkurencję „wygra" inny neuron. Nie będzie w tym zresztą nic zaskakującego, bo mapa pokazująca rozmieszczenie wartości współczynników wagowych pozwoli łatwo ustalić, który neuron będzie zwycięzcą, gdy pokaże się taki lub inny sygnał wejściowy – po prostu będzie to ten neuron, którego wektor współczynników wagowych znajdzie się najbliżej wektora reprezentującego zaprezentowany sygnał wejściowy.

Z faktem, że pewien neuron został zwycięzcą, związanych jest kilka konsekwencji. Po pierwsze, w większości sieci tego typu **tylko ten jeden neuron ma niezerową wartość sygnału wyjściowego** (zwykle wartość ta ustalana jest jako równa **1**). Sygnały wyjściowe wszystkich pozostałych neuronów sieci są sztucznie zerowane, co w literaturze określane jest często skrótem WTA (*Winner Takes All* – zwycięzca bierze wszystko).

Nie dość jednak na tym – również **proces uczenia** (dokładniej – samouczenia) **ogranicza się zwykle do samego tylko „zwycięzcy".** Jego (i tylko jego!) współczynniki wagowe są zmieniane, przy czym proces ten przebiega w taki sposób i w takim kierunku, by przy następnym konkursie z tym samym sygnałem wejściowym ten „zwycięski" neuron wygrał jeszcze bardziej przekonywująco!

Jaki to ma sens?

W celu odpowiedzi na to pytanie prześledźmy dokładnie, co się dzieje w takiej samouczącej się sieci z konkurencją. Pojawia się oto na wejściu sieci obiekt, reprezentowany przez swoje sygnały wejściowe. Sygnały te są podawane do wszystkich neuronów i tworzone są ich „wypadkowe pobudzenia". W najprostszym przypadku jest ono obliczane na zasadzie przemnożenia sygnałów wejściowych przez wagi danego neuronu, ale można sobie wyobrazić tę samą zasadę z wykorzystaniem na przykład neuronów o nieliniowych charakterystykach. Im bardziej wagi neuronu są podobne do sygnałów wejściowych – tym silniejsze powstaje „wypadkowe pobudzenie" na wyjściu tego właśnie (każdego kolejno rozważanego) neuronu. Mówiliśmy już o tym, że zestawy wag można traktować jako „wzorce" sygnałów wejściowych, na które każdy neuron jest uwrażliwiony. Zatem im bardziej sygnał wejściowy przypomina wzorzec przechowywany przez rozważany neuron – tym silniejszym sygnałem odpowiada on na jego pojawienie się. Jeśli więc któryś neuron zostaje w pewnym momencie „zwycięzcą" – to będzie znaczyło, że jego „wzorzec wewnętrzny" jest najbardziej podobny do aktualnie pojawiającego się sygnału wejściowego.

Ale z jakiego powodu ma być podobny?

Otóż na początku uczenia może to wynikać wyłącznie z losowego procesu inicjalizacji wag. W każdej sieci neuronowej **początkowe** wartości

współczynników wag nadawane są wszystkim neuronom w sposób losowy. Te przypadkowo przydzielone współczynniki wagowe okazują się potem bardziej lub mniej podobne do tych zestawów sygnałów wejściowych, które sieć otrzymuje podczas uczenia, jako kolejne, rozpoznawane przez nią obiekty. Niektóre neurony mają zatem – na zasadzie czystego przypadku – „wrodzoną skłonność" do rozpoznawania pewnych obiektów oraz – równie przypadkową „niechęć" do rozpoznawania innych. Potem, w miarę jak postęp uczenia wymusza wzrastające z kroku na krok podobieństwo wewnętrznych wzorców przechowywanych w neuronach sieci do pewnych typowych obiektów, pojawiających się szczególnie często w procesie uczenia – przypadkowość znika i neurony coraz dokładniej specjalizują się w rozpoznawaniu przynależnych im klas.

Na tym etapie neuron, który raz „zwyciężył" podczas próby rozpoznawania litery **A** – z tym większym prawdopodobieństwem zwycięży po raz kolejny, gdy na wejściu sieci pojawi się inna litera **A**, nawet napisana inną ręką. Na początku jest jednak zawsze sytuacja losowa – neurony same decydują, które z nich mają rozpoznawać **A,** które **B**, a które sygnalizować, że pokazany znak nie jest wcale żadną literą, tylko – na przykład odciskiem brudnego palucha. Proces samouczenia zawsze jedynie te **naturalne skłonności** (powtarzam: przypadkowo powstające w wyniku wstępnej generacji wag) wzmacnia i krystalizuje.

Tak się jednak dzieje w każdej sieci z samouczeniem, jaki wobec tego wpływ na te procesy ma zjawisko konkurencji?

Otóż **w wyniku konkurencji proces samouczenia może być bardziej skuteczny i ekonomiczny.**

Jeśli początkowe wartości współczynników wag neuronów są losowe, to może się zdarzyć, że czasem kilka neuronów wykazuje „skłonność" do rozpoznawania tej samej klasy obiektów. Normalny, to znaczy pozbawiony elementu konkurencji proces samouczenia będzie te „skłonności" rozwijał współbieżnie we wszystkich tych neuronach, nie będzie zatem dochodziło do różnicowania zachowania poszczególnych elementów sieci, a przeciwnie – będzie postępował proces rosnącej uniformizacji. Widziałeś to dokładnie podczas licznych opisanych wyżej eksperymentów z programem **Example 10b.**

Jeśli jednak wprowadzimy konkurencję, to sytuacja ulegnie zmianie. Zawsze wtedy któryś neuron będzie chociaż odrobinę **bardziej** dostosowany do rozpoznawania aktualnie pokazywanego obiektu niż jego „konkurenci". Naturalną konsekwencją tego będzie fakt, że taki właśnie neuron, mający przypadkowo współczynniki wag podobne do cech aktualnie prezentowanego obiektu, zostanie oczywiście „zwycięzcą". Zastosowanie do tego (i tylko tego) neuronu procesu uczenia spowoduje, że jego „wrodzona skłonność"

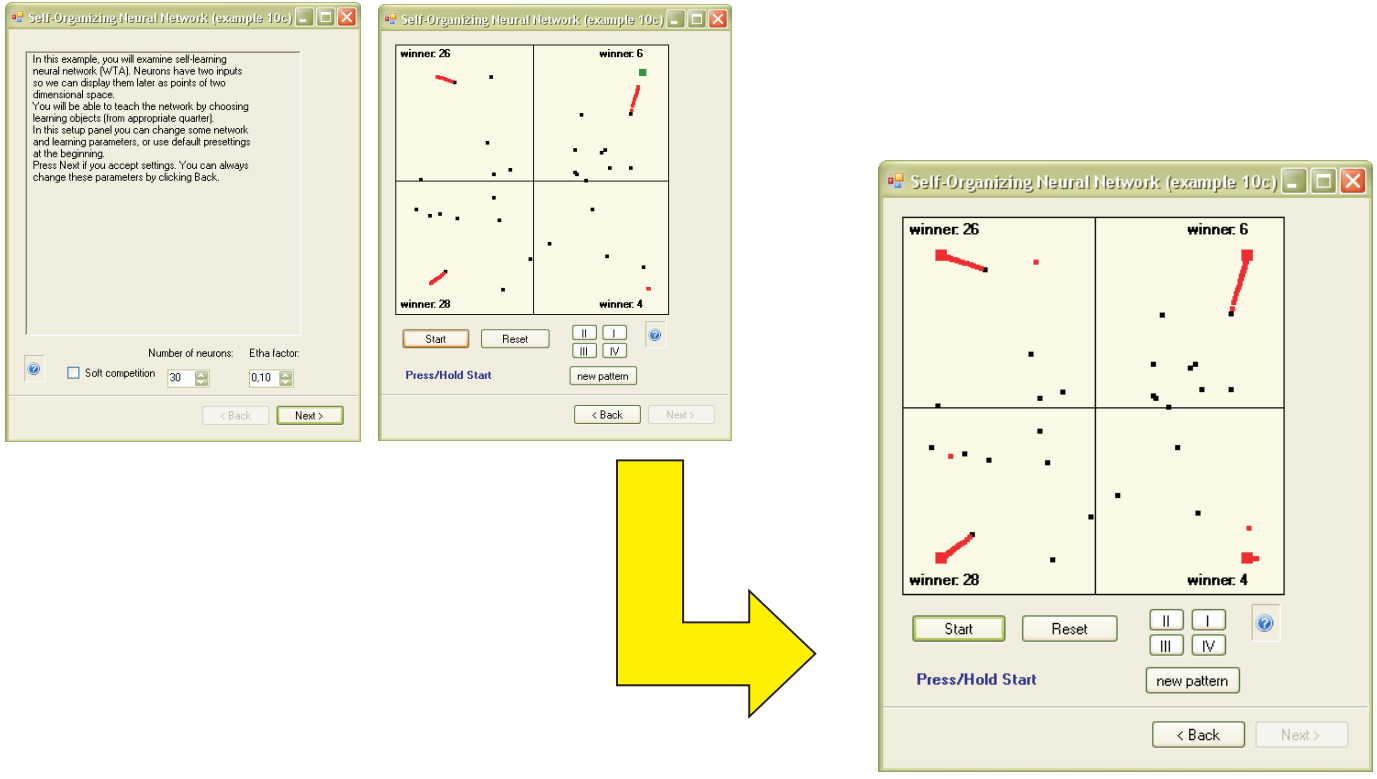
zostanie – w miarę rozwoju procesu uczenia – wzmocniona i skrystalizowana, a „konkurencja" zostanie z tyłu i będzie mogła ewentualnie rywalizować o rozpoznawanie innych, jeszcze „nie obstawionych" obiektów.

Przebieg samouczenia z konkurencją najlepiej prześledzisz korzystając z programu **Example 10c.** W programie tym zastosowałem uczenie z konkurencją do podobnego zadania (nauki rozpoznawania Marsjan, Marsjanek, Marsjaniątek i Marsjamorów), co w poprzednich dwóch programach przedstawionych w tym samym rozdziale. Ponieważ jednak tym razem zasada uczenia jest odmienna – zobaczysz w trakcie obserwacji zachowania sieci istotnie odmienne zjawiska.

Najpierw program tak jak w poprzednich przykładach, wyświetli okno z parametrami, gdzie m.in. możesz ustawić liczbę neuronów w uczącej się sieci. Na początku proponuję Ci zmianę tylko tego parametru, bo jak zaczniesz manipulować równocześnie wszystkimi parametrami – to się w tym całkiem pogubisz.

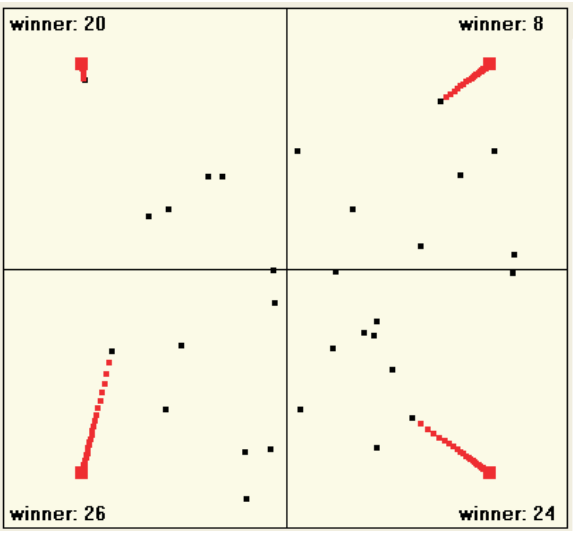
Zasady wyboru liczby neuronów są dosyć proste: Przy mniejszej liczbie neuronów (na przykład 30) zjawiska samouczenia z konkurencją wychodzą efektowniej. Możesz wtedy wyraźnie zaobserwować, że wszystkie neurony „stoją", a tylko zwycięzca podlega uczeniu. Dla uwidocznienia tego faktu przewidziałem, że na ekranie pozostawać będzie ślad kolejnych położeń uczącego się neuronu, co wygodnie pozwala śledzić jego „trajektorię". Na rysunkach produkowanych przez program **Example 10c** przyjęto dodatkowo taką zasadę, że w momencie, kiedy neuron osiąga już swoje docelowe położenie – punkt wskazywany przez zestaw jego współczynników pojawia się wtedy w miejscu pokazywanego wzorca jako duży czerwony kwadrat. Możesz to uważać za sygnał zakończenia procesu samouczenia – przynajmniej dla tej jednej klasy (rys. 9.37).

Po uruchomieniu procesu samouczenia zobaczysz, że do każdego z punktów, w których pojawiać się będą obiekty należące do prezentowanych sieci klas, „przyciągnięty" zostanie **pojedynczy** neuron, który po pewnym czasie stanie się idealnym detektorem obiektów pochodzących z tej właśnie klasy (zobaczysz to na ekranie w formie pojawienia się dużego czerwonego kwadratu w miejscu, gdzie typowo pojawiały się obiekty tej właśnie klasy). Naciskając kolejno przycisk **Start** proces samouczenia wykonuje się tak jak w poprzednim przykładzie „krok po kroku" (jeśli klikniesz **Start** i nie zwolnisz kliknięcia – proces samouczenia stanie się automatyczny). Śledząc trajektorie przemieszczających się wektorów wag poszczególnych neuronów (a także komunikat wyświetlany w każdej ćwiartce informujący o tym, który neuron stał się w danej chwili „zwycięzcą") możesz zobaczyć, że w każdej ćwiartce w toku tego etapu procesu samouczenia znajdowany jest jeden neuron, który



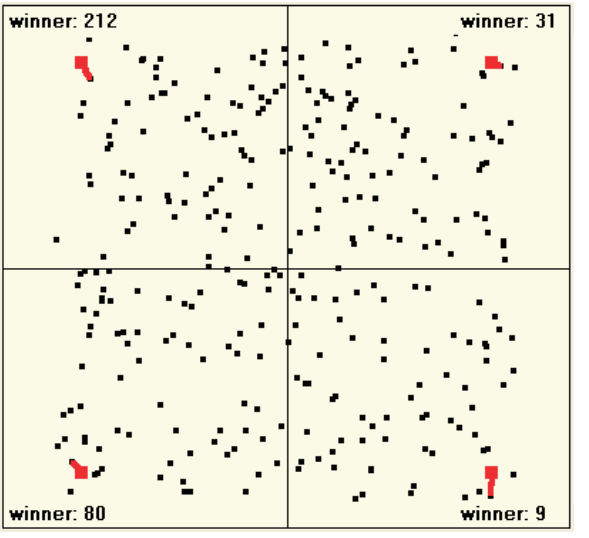

Rys. 9.37. Okno z parametrami programu Example 10c oraz sposób prezentacji przebiegu samouczenia w sieci z konkurencją – przed i po osiągnięciu końcowego sukcesu

w wyniku prezentacji kolejnych przykładów z danej ćwiartki stale wygrywa i dlatego tylko on zmienia położenie w kierunku pokazywanego wzorca, aż wreszcie odnajduje swoje miejsce i wtedy nieruchomieje (rys. 9.38).

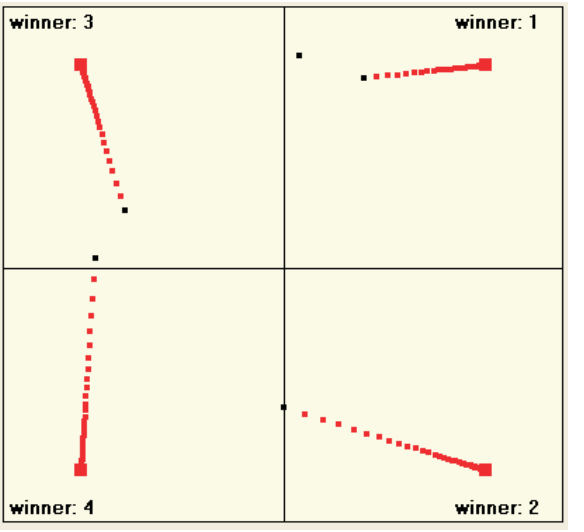



Rys. 9.38. Przebieg samouczenia w sieci z konkurencją

Przy dużej liczbie neuronów proces samouczenia z konkurencją jest trudniej obserwować, bo ma on (w odróżnieniu od klasycznego samouczenia dużych sieci) bardzo lokalny charakter (rys. 9.39). Wynika to z faktu, że przy




Rys. 9.39. Proces samouczenia z konkurencją w dużej sieci




Rys. 9.40. Proces samouczenia w sieci z małą liczbą neuronów

licznych neuronach gęsto rozrzuconych dystans od najbliższego sąsiada (neuronu najbliższego w stosunku do pokazanego wzorca) jest bardzo mały i trajektoria nauczanego „zwycięzcy" jest w wyniku tego mało zauważalna.

Z kolei przy bardzo małej liczbie neuronów (na przykład 5) trajektorie będą efektowne i długie, a w dodatku będziesz mógł zauważyć, że przy zastosowaniu konkurencji nawet bardzo słaba początkowo skłonność jakiegoś neuronu do rozpoznawania jakiej klasy obiektów może zostać wychwycona i wzmocniona przez proces uczenia – pod warunkiem że „konkurenci" będą mieli jeszcze słabszą skłonność do rozpoznawania tej właśnie klasy obiektów (rys. 9.40). Dzięki obecności na ekranie śladu kolejnych położeń uczących się neuronów zauważysz też jeszcze jedną, dość interesującą i ogólną cechę sieci neuronowych – mianowicie to, że postęp uczenia na początku tego procesu jest szybszy i większy niż w jego końcowej fazie.

## 9.9. Jakie formy samouczenia daje wprowadzenie konkurencji do sieci?

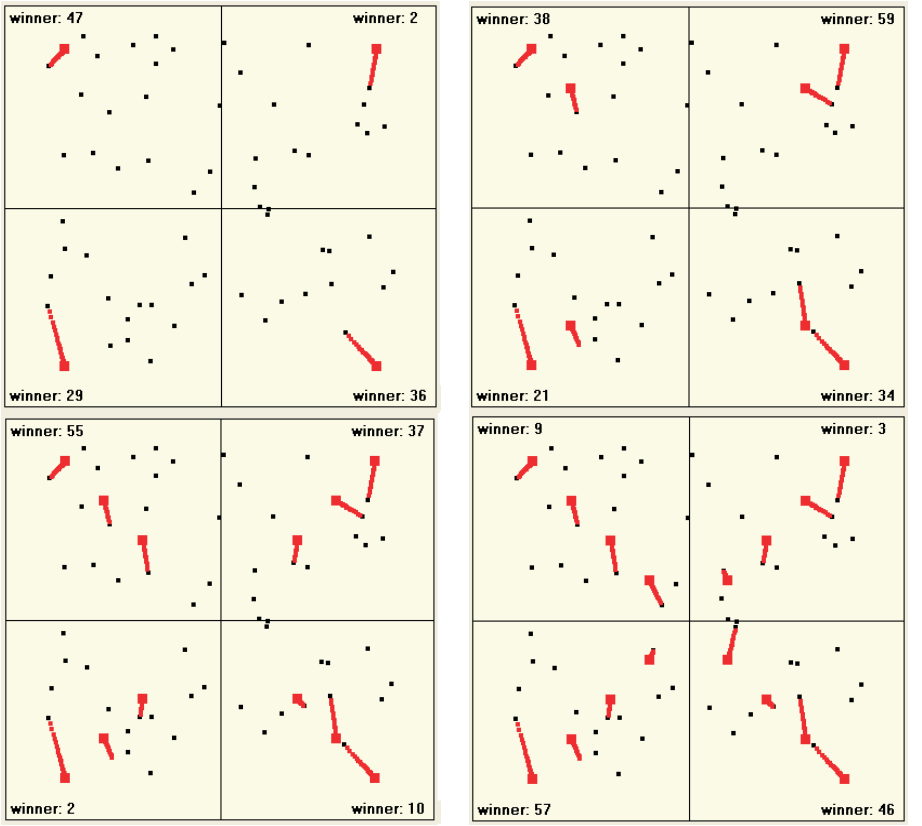
W Programie **Example 10c** zaraz po uruchomieniu (w oknie z parametrami – rys. 9.37 po lewej stronie) możesz również ustawić parametr **Soft competition** odpowiedzialny za „złagodzoną" konkurencję. Na początku sugeruję, byś z tej opcji nie korzystał (wystarczy, że zostawisz to pole tak jak jest to domyślnie, puste). Do eksperymentów ze złagodzoną konkurencją przystąpimy nieco później, gdy już poznasz zalety, ale także i mankamenty konkurencji „twardej".

Dzięki „twardej" konkurencji w programie **Example 10c** nie będą się tworzyły znane Ci z eksperymentów z programami **Example 10a** i **Example 10b** „kliki" neuronów mających dokładnie te same preferencje. Przy odrobinie szczęścia (zwłaszcza w sieciach zawierających spory nadmiar neuronów w stosunku do liczby klas) nie powinny się także tworzyć luki i „martwe pola" w reprezentacji wejściowych obiektów przez poszczególne neurony, czyli obiekty, do których nikt się nie chce przyznać w procesie rozpoznawania. Po zastosowaniu konkurencji można więc z dużym prawdopodobieństwem przyjąć, że nie będzie w sieci grup neuronów rozpoznających tę samą klasę i nie będzie klas, które nie będą rozpoznawane przez żaden neuron. Zupełnie jak w znanym polskim przysłowiu: „*każda potwora znajdzie swego amatora*[3]".

Dzięki temu, że podczas uczenia z konkurencją inne neurony (poza zwycięzcą) nie zmieniają swojego położenia, są one gotowe do przyjęcia i zaakceptowania innego wzorca. Z tego powodu w sytuacji, gdy podczas uczenia pojawią się nagle obiekty z zupełnie innych klas – w sieci **będą** nadal wolne neurony gotowe do tego, by niezwłocznie podjąć trud rozpoznawania tych nowych klas i systematycznego doskonalenia się w ich identyfikowaniu. Możesz to doskonale prześledzić na swoim ekranie, ponieważ w programie **Example 10c** wbudowałem możliwość zażądania, by w polu widzenia już nauczonej sieci pojawiły się nowe, do tej pory całkiem nieznane obiekty. Efekt ten uzyskasz, naciskając w dowolnym momencie podczas symulacji przycisk **new pattern**. Naciskając kilka razy przycisk **new pattern** i obserwując wyniki możesz doprowadzić do tego, że Twoja sieć uzyska w trakcie samouczenia umiejętność rozpoznawania znacznie większej liczby klas niż stale do tej pory eksploatowane cztery typy Marsjan (nie żądaj jednak ode mnie, żebym Ci powiedział, co też te inne klasy mogą przedstawiać, bo po wymyśleniu Marsjamorów moja fantazja została wyczerpana!). Dzięki samouczeniu z konkurencją każda z tych licznych klas wejściowych obiektów zyska sobie „anioła stróża" w postaci neuronu, który będzie się od tej pory z tą właśnie klasą utożsamiał (rys. 9.41) – i co więcej, na ogół zostanie jeszcze trochę swobodnych (nie zagospodarowanych) neuronów do dalszego rozpoznawania wzorców, które być może pojawią się w przyszłości.

Niestety, czysty proces samouczenia połączony z „twardą" konkurencją także może prowadzić do pewnych wynaturzeń. Kiedy uruchomisz proces samouczenia z niewielką liczbą neuronów – dla każdej z pokazywanych (czterech w moim programie) nielicznych na początku klas znajdzie się neuron,

[3] Wszak wiesz, że gdyby nie konkurencja typu *WTA* (wynikająca z obowiązującej w naszym kraju biologicznie nie uzasadnionej monogamii) wszystkie dziewczyny przypadłyby w udziale wyłącznie intelektualnej elicie społeczeństwa (to znaczy informatykom), a tak – to od czasu do czasu ktoś inny też się załapie ...

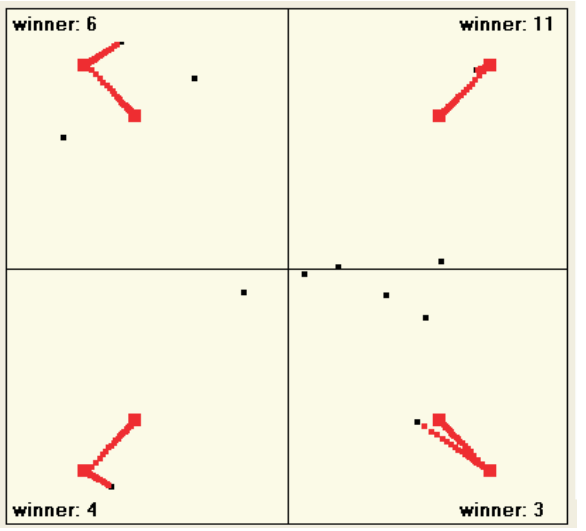



Rys. 9.41. Możliwość uczenia sieci z konkurencją rozpoznawania wielu wzorców

który będzie daną klasę sygnalizował czy wykrywał. Jeśli jednak pojawią się nowe klasy obiektów (po kliknięciu przycisku **new pattern**) może się zdarzyć, że „zwycięzcą" przy rozpoznawaniu którejś z nowych klas stanie się neuron, który uprzednio wyspecjalizował się w rozpoznawaniu innej klasy! Może to powodować, że pewna klasa obiektów wejściowych, wcześniej już wytrenowana i „wyposażona" w neuronową reprezentację – nagle ją utraci[4]! Oznacza to, że w sieci występuje znany Ci zresztą z codziennego doświadczenia fakt, polegający na tym, że nowe nabywane wiadomości „wypierają" stare.

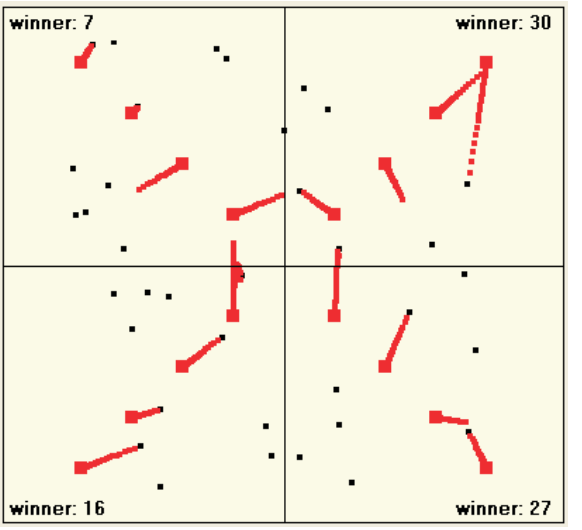
Na rysunku 9.42 pokazano opisany efekt na przykładzie, kiedy przybiera on formę wręcz krańcową – **wszystkie** wcześniej wyuczone neurony, rozpoznające pewne obiekty, zostały po pokazaniu nowych obiektów „przeuczone" do tego, by wykrywać te właśnie nowe klasy. Nie jest to sytuacja typowa.

[4] Pozostając nadal przy żartobliwych analogiach matrymonialnej natury można powiedzieć, że obserwujemy oto mechanizm powstawania zdrady małżeńskiej oraz rozwodów.




Rys. 9.42. Efekt wypierania przez nowe wiadomości pamięci dawniej wyuczonych umiejętności

Zwykle neuronów jest na tyle dużo w stosunku do liczby klas, że efekt „kidnapingu" występuje raczej rzadko – co najwyżej w przypadku jednej czy dwóch klas na kilkanaście (rys. 9.43). Tłumaczy to, dlaczego w naszej codziennej praktyce związanej z użytkowaniem własnej sieci neuronowej, czyli naszego własnego mózgu – tylko niektóre, na szczęście raczej nieliczne wyuczone wiadomości giną nagle z pamięci, wyparte przez inne, intensywnie zapamiętane wrażenia. Niestety pech polega na tym, że zwykle „giną" z naszej pamię-

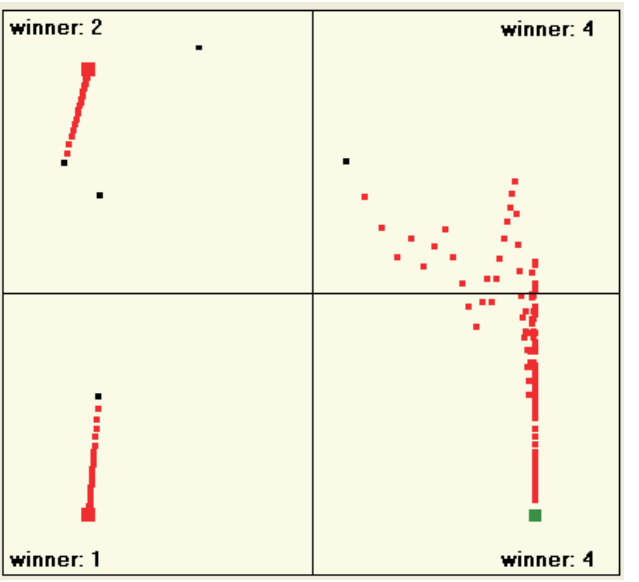



Rys. 9.43. Efekt gubienia tylko drobnej części wcześniej wyuczonych wiadomości w sieci z „twardą" konkurencją

ci ślady najbardziej istotnych wiadomości – i to zwłaszcza wtedy, gdy się ich najbardziej potrzebuje!

Podczas eksperymentów z programem **Example 10c** z pewnością sam dostrzeżesz dalsze uwarunkowania zjawiska „porywania" przez nowe obiekty neuronów znajdujących się już w ustabilizowanym wcześniej związku z innymi klasami, przy czym – jak widać z przytoczonych rysunków – może się to zdarzyć również w takim przypadku, gdy jest jeszcze sporo neuronów nie mających ustalonych związków z pokazywanymi obiektami – ale położonych dalej od obszarów, w których pojawiają się prezentowane sieci wzorce. W przykładowym programie zjawisko to nasila się na skutek pokazywania najpierw jednej grupy obiektów, a potem, gdy już są ustalone przyporządkowania neuronów do tej pierwszej grupy – nowych obiektów należących do nowych klas, które czasami bezkarnie odbierają neurony z tych wcześniej ustalonych klas. Gdyby wszystkie obiekty pokazywały się równocześnie – dochodziłoby do swoistej rywalizacji, objawiającej się „przeciąganiem" neuronów raz w kierunku jednej, potem zaś w kierunku innej klasy (możesz to zaobserwować także w programie **Example 10c**, uruchamiając proces samouczenia z bardzo małą liczbą neuronów – patrz rys. 9.44).

W takim przypadku jest możliwe, że sieć **niczego** się nie nauczy mimo długiego czasu samouczenia. To także jest Ci znane z codziennych doświadczeń, przypomnij sobie chociażby kłopoty szkolne, kiedy mając na drugi dzień liczne sprawdziany – na przykład z matematyki, fizyki, języka, geogra-

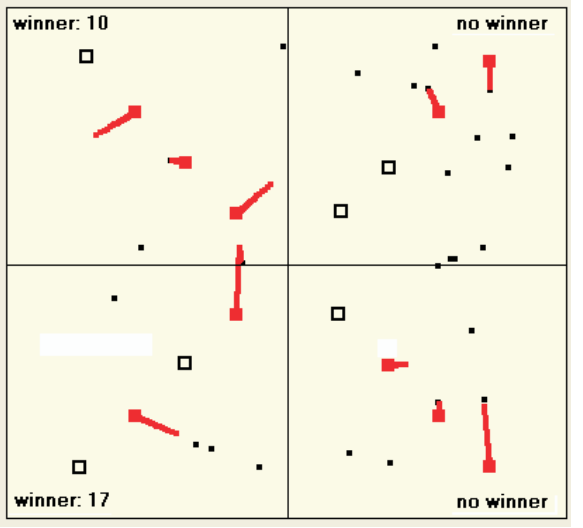



Rys. 9.44. Efekt „przeciągania" neuronu pomiędzy obiektami klasy 1. i klasy 4.

fii i historii – nie byłeś w stanie przygotować się dobrze do **żadnego** z tych przedmiotów i w rezultacie „oblewałeś" wszystkie egzaminy!

Radą na opisane wyżej kłopoty może być złagodzenie konkurencji. Możesz tego dokonać wracając do okna z parametrami (przyciskiem **Back**). Okno to pokazałem Ci na rysunku 9.37 po lewej stronie. Odnajdując w tym oknie i ustawiając (zaznaczając jako wypełniony *checkbox*) parametr **Soft competition** możesz zażądać złagodzonej konkurencji jako reguły działania w Twojej sieci. Od tej pory program stosować będzie konkurencję ograniczoną, to znaczny neuron zwycięzca będzie wyłaniany tylko wtedy, jeśli „wylegitymuje się" naprawdę dużą wartością wyjściowego sygnału[5]. Konsekwentne stosowanie tej zasady spowoduje, że neurony będą bardziej równomiernie rozdzielone między klasy obiektów wymagające rozpoznawania. W wyniku zastosowania złagodzonej konkurencji nie będą też miały miejsca gorszące sceny wydzierania przez nowe klasy neuronów już wcześniej zaanektowanych przez inne klasy.

Niestety – zamiast tego zauważysz inne martwiące zjawisko: w pewnych okolicznościach (zwłaszcza przy niewielkiej liczbie neuronów) może dochodzić do sytuacji, że po pokazaniu kolejnego obiektu **żaden** neuron nie uzyska statusu zwycięzcy[6] (rys. 9.45).




Rys. 9.45. Omijanie niektórych klas przy złagodzonej konkurencji

[5] Nadal bawiąc się analogiami pochodzącymi z obserwacji stosunków międzyludzkich można powiedzieć, że złagodzona konkurencja odpowiada sytuacji, w której partnerzy muszą się bardzo mocno zakochać, żeby mogli zmierzać do małżeństwa. Słaba i przejściowa fascynacja nie prowadzi do żadnych trwałych konsekwencji.

[6] Dla tego efektu także łatwo znajdziesz analogię wywodzącą się ze stosunków międzyludzkich: Z całą pewnością znasz w swoim otoczeniu „stare panny" oraz „wiecznych kawalerów". Ich obecność to jest właśnie efekt połączenia zasady **konkurencji** (monogamia) oraz stosowaniea jej **słabej wersji** (konieczne jest bardzo silne uczucie, żeby doszło do małżeństwa).

Używając programu **Example 10c** przy złagodzonej konkurencji będziesz mógł to zjawisko dokładnie przebadać i przeanalizować. Będzie to tym łatwiejsze, że program w tym trybie pracy stosuje specjalne markery, oznaczające położenie wzorców pomijanych klas (to znaczy takich klas, dla których żaden neuron nie okazał się zwycięzcą).

Przekonasz się, że przy braku ostrej konkurencji często będziesz się spotykać z sytuacją takich właśnie pominięć, zwłaszcza wtedy, gdy modelować będziesz małe sieci (o niewielkiej liczbie neuronów). W takich przypadkach dla większości klas w sieci wytworzą się – całkowicie samorzutnie! – wzorce pozwalające na późniejsze automatyczne rozpoznawanie odpowiednich klas sygnałów, ale dla „pechowej" klasy, której żaden neuron nie chce sygnalizować – **w ogóle** nie wytworzą się w sieci wyspecjalizowane detektory.

– I co?

– I nic!

Czy nigdy nie widziałeś sytuacji, że ktoś świetnie uczy się historii, geografii, języków – a matematyka w żaden sposób „nie wchodzi mu do głowy"?

## 9.10. Pytania kontrolne oraz zadania do samodzielnego rozwiązania

1. Spróbuj uzasadnić, komu, kiedy i do czego może być potrzebne samouczenie sieci neuronowej.
2. Jaki czynnik decyduje o tym, które punkty reprezentujące neurony w programie **Example 10a** będą miały na ekranie barwę czerwoną, a które – niebieską?
3. Na podstawie opisanych w książce, a także przeprowadzonych przez Ciebie, obserwacji i eksperymentów, spróbuj sformułować własne wnioski na temat wpływu, jaki na zdobytą wiedzę mają wrodzone talenty (reprezentowane przez początkowy przypadkowy rozkład wektorów wag neuronów) oraz doświadczenie życiowe (reprezentowane przez proces samouczenia). Czy Twoje obserwacje skłaniają Cię do przyjęcia poglądu, że systemy mające małą liczbę neuronów (prymitywne zwierzęta) są silniej zdeterminowane przez swoje instynkty i właściwości wrodzone, natomiast systemy bardziej złożone, wyposażone w dużą liczbę neuronów (umysł człowieka) znajdują się pod silniejszym wpływem czynników związanych z doświadczeniem osobistym i celowym kształceniem? Spróbuj zebrać konkretne argumenty przemawiające za tą tezą (lub przeciwko niej) w postaci wyników samouczenia sieci o różnych rozmiarach.
4. Omów właściwości sieci samouczących się przy szczególnie małych i przy szczególnie dużych wartościach współczynnika uczenia ***Etha.*** Zasta-

nów się, czy podobne zjawiska występują również w mózgach zwierząt? Przy rozważaniu tej kwestii weź pod uwagę okoliczność, że podatność mózgu na uczenie (będąca odpowiednikiem współczynnika uczenia ***Etha***) maleje z wiekiem, to znaczy na przykład młody szczeniak dysponujący mózgiem, którego modelem może być sieć o dużej wartości ***Etha***, zachowuje się inaczej niż dorosły pies (którego ***Etha*** czasem zbliża się wręcz do zera). Spróbuj wskazać dobre i złe skutki tego zjawiska.

5. Czy samoucząca się sieć neuronowa, której zdolności do „fantazjowania" opisano w tym rozdziale, potrafi wymyślić jakieś całkiem nowe pojęcie, nie mające żadnego odpowiednika w rzeczywistym świecie, a dokładniej – w przestrzeni sygnałów wejściowych, z których pochodzą sygnały na podstawie których sieć dokonuje samouczenia?

6. Czy możesz wyobrazić sobie rzeczywistą sytuację odpowiadającą doświadczeniu opisanemu w podrozdziale 9.7? Spróbuj opisać taką sytuację i zastanów się, jaki miałaby ona wpływ na funkcjonowanie mózgu doświadczalnego zwierzęcia albo umysłu poddanego takim zabiegom człowieka (zwłaszcza małego dziecka).

7. Czy Twoim zdaniem mechanizm **zapominania** powstającego w wyniku „przeuczenia" neuronów, który został opisany w treści podrozdziału 9.6, tłumaczy wszystkie fenomeny związane z tym uciążliwym (zwłaszcza przed egzaminem…) procesem? A niezależnie od mechanizmu – czy zapominanie wiadomości, które nie są stale aktualizowane poprzez pojawianie się zadań, w których te wiadomości trzeba wykorzystywać – jest biologicznie korzystne, czy niekorzystne?

8. Jakie niekorzystne zjawiska, występujące podczas samouczenia sieci, dają się wyeliminować poprzez wprowadzenie mechanizmu konkurencji, a które mimo wszystko nadal pozostają?

9. W sieciach z konkurencją poza omówioną w rozdziale zasadą WTA (*Winner Takes All*) używana jest niekiedy zasada WTM (*Winner Takes Most*). Zastanów się, co to może znaczyć, a potem sprawdź trafność Twoich przypuszczeń w Internecie.

10. Załóżmy, że chcesz zbudować sieć samoucząca się, która będzie w stanie skutecznie zbudować wzorce dla ośmiu klas. Spróbuj eksperymentalnie ustalić, o ile więcej neuronów (od minimalnej ich liczby wynoszącej 8) powinna mieć taka sieć na początku procesu samouczenia, żeby nie dochodziło w niej do pomijania żadnej z klas, a jednocześnie żeby liczba „nie zagospodarowanych" neuronów po procesie samouczenia nie była zbyt duża.

11. **Zadanie dla zaawansowanych.** Zbuduj wersję programu **Example 10b**, w którym byłaby możliwość zaprogramowania z góry zmienności parametru ***Etha*** (w funkcji liczby pokazanych do tego momentu danych ze zbioru

uczącego) oraz w którym można by było zaplanować pojawianie się pewnych obiektów tylko w pewnym okresie „życia" sieci (na przykład niektóre obiekty są widziane tylko w „dzieciństwie" bez kontynuacji w „życiu dojrzałym", a inne pojawiają się dopiero w późniejszych etapach życia i doświadczenia uczącego się „osobnika"). Przeprowadź serię eksperymentów z takim bardziej „życiowym" programem, a następnie spisz spostrzeżenia i postaraj się wyciągnąć wszystkie możliwe wnioski. W jakim zakresie zjawiska obserwowane w sieci mają swoje odpowiedniki w życiu, a w jakim są wynikiem wyłącznie tego, że budowana i badana sieć jest bardzo daleko idącym uproszczeniem rzeczywistych struktur nerwowych występujących w mózgu człowieka i zwierząt?

12. **Zadanie dla zaawansowanych.** Zbuduj wersję programu **Example 10c**, w którym byłaby uwzględniona konkurencja z wykorzystaniem czynnika „sumienia". Zastosowanie „sumienia" oznacza, że neuron, który często wygrywa, w miarę upływu czasu uczenia coraz częściej „rezygnuje z wygranej" i pozwala wygrywać innym neuronom. Porównaj zachowanie takiego modelu sieci „z konkurencją i sumieniem" oraz modelu z „czystą, ostrą konkurencją". Jakie wnioski możesz wyciągnąć? Czy wnioski te odnoszą się tylko do dziedziny sieci neuronowych, czy też można je jakoś odnieść także do zjawisk i procesów (na przykład gospodarczych) zachodzących w rzeczywistym świecie?

# 10. Sieci samoorganizujące się

## 10.1. Jaką strukturę ma sieć neuronowa, w której tworzyć będziesz odwzorowania, będące skutkiem samoorganizacji?

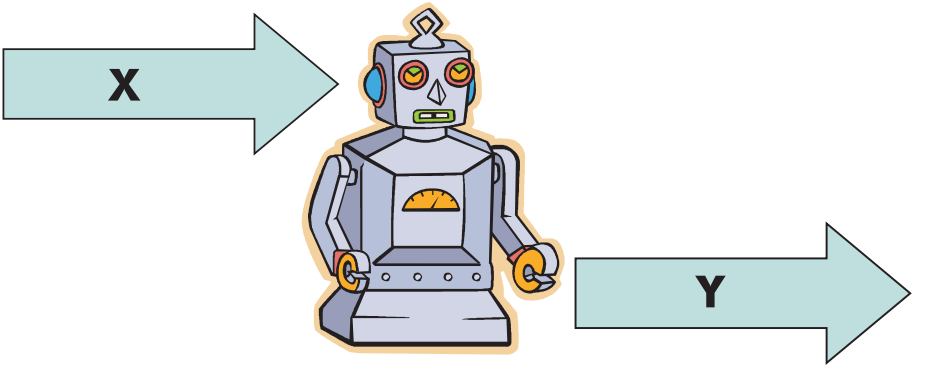
Sieć neuronowa może być nauczana z nauczycielem lub może się uczyć sama. O tym już wiesz, to już przerabiałeś. Teraz natomiast poznasz sieci, które nie tylko same się uczą (bez nauczyciela), ale w dodatku tak uzgadniają (na zasadzie wzajemnych oddziaływań) funkcjonowanie wszystkich neuronów, że ich łączne, wypadkowe działanie stwarza pewną nową jakość. Chodzi o automatycznie tworzone przez sieć złożone **odwzorowanie** (po angielsku *mapping*) zbioru sygnałów wejściowych w zbiór sygnałów wyjściowych. Odwzorowanie to na ogół nie daje się z góry przewidzieć ani jakoś specjalnie zdeterminować, więc omawiane tu sieci są znacznie bardziej samodzielne i niezależne niż sieci, z którymi miałeś wcześniej do czynienia, ponieważ ich właściwości są tylko w niewielkim stopniu narzucone przez twórcę sieci, natomiast istotę ich działania oraz końcowe odwzorowanie, będące skutkiem ich pracy – determinuje głównie wspomniany proces samoorganizacji. Szczegóły tego procesu unaoczni Ci już za chwilę kolejny przygotowany program. Zanim to jednak nastąpi, chciałem Ci w skrócie powiedzieć, do czego to wszystko może służyć, żebyś nie ograniczał swoich zainteresowań do samego tylko oglądania obrazków, ale spróbował sobie wyobrazić sieci samoorganizujące się jako poważne narzędzia służące do poważnych celów.

Odwzorowanie jest dość bogatym i skomplikowanym pojęciem matematycznym, mającym – zależnie od okoliczności – różne konkretne cechy i właściwości. Nie chcę Cię tutaj zanudzać informacjami na temat teorii, więc spróbuję maksymalnie poglądowo (ale niestety, z konieczności niezbyt ściśle) omówić samą istotę rzeczy, nie wnikając w ważne, ale trudne i skomplikowane szczegóły. Nie wiem, czy mi się to uda, więc umówmy się – gdybyś pod-

czas czytania tych moich wyjaśnień doszedł do wniosku, że jest to jednak za trudne (albo za nudne...) – to możesz bez żalu porzucić całą tą teorię i zacząć czytać od następnego podrozdziału już konkretne i szczegółowe informacje, jak samemu zbudować sieć samoorganizującą się i jak badać jej właściwości. Jeśli jednak zdobędziesz się na trud przeczytania tego teoretycznego wprowadzenia – to dowiesz się, po co takie samoorganizujące sieci się buduje. Można bez tego przeżyć, ale będziesz miał więcej satysfakcji czytając potem i oglądając, co taka sieć robi, wiedząc również – do czego to się może przydać.

Zakładam, że jeśli to czytasz, to jesteś żądnym wiedzy ryzykantem i mogę Cię wprowadzić w sprawy trochę trudniejsze. Pamiętaj – sam tego chciałeś!

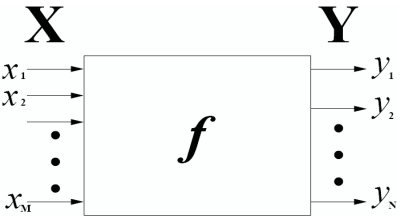
Zacznijmy od stwierdzenia, że generalnie bardzo często potrzebujemy przekształcać sygnały wejściowe w sygnały wyjściowe według pewnych reguł. Na przykład budując roboty, które mają spełniać określone zadania, musimy zapewnić, że ich systemy sterujące będą mogły w prawidłowy sposób przekształcać sygnały odbierane przez ich sensory (kamery video, mikrofony, wyłączniki kontaktowe zastępujące dotyk, ultradźwiękowe czujniki zbliżenia itp.) na sygnały sterujące pracą mechanizmów napędowych nóg, chwytników, elementów ramion itp. (rys. 10.1).




Rys. 10.1. Funkcjonowanie robota polega na przekształcaniu sygnałów X w działania Y

Właśnie takie przekształcanie dowolnych sygnałów wejściowych we właściwe sygnały wyjściowe jest **odwzorowaniem** (rys. 10.2).

Oczywiście, nie może to być jakieś dowolne odwzorowanie, bo nie będzie nam do niczego przydatne. Żeby robot poprawnie poruszał się, wykonywał sensowne zadania, poprawnie reagował na polecenia – odwzorowanie wiążące bodźce rejestrowane za pomocą sensorów z określonymi ruchami wykonywanymi za pomocą elementów wykonawczych musi być odpowiednio



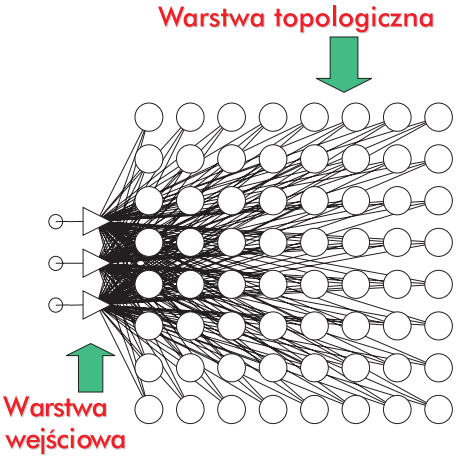

Rys. 10.2 Ogólne przedstawienie **odwzorowania**

dobrane, szczegółowo zaprogramowane, dokładnie przemyślane. Konstruktor robota (podobnie jak konstruktorzy wielu innych systemów automatyki) stoi więc przed trudnym zadaniem określenia pożądanego odwzorowania, będącego w istocie zasadniczą zawartością „mózgu" (sterownika) robota.

Jeśli sygnał wejściowy jest jeden i sygnał wyjściowy jest jeden – to zadanie jest w miarę proste, a potrzebne odwzorowanie jest znaną Ci zapewne ze szkoły **funkcją.** Jeśli jednak sensorów stanowiących zmysły robota jest dużo (a w takich przypadkach **zawsze** jest ich dużo!) oraz jeśli elementów wykonawczych też jest dużo (a musi ich być dużo, jeśli robot ma coś ciekawego zrobić) – to zadanie staje się bardzo trudne, kłopotliwe i czasochłonne. Można wprawdzie „ręcznie" ustawić i opisać wszystkie potrzebne odwzorowania, ale przy dużych i skomplikowanych zadaniach może to zająć całą resztę życia.

W takim przypadku pomocne mogą się okazać właśnie samoorganizujące się sieci neuronowe. Przykładowa struktura takiej sieci przedstawiona jest na rysunku 10.3.

Sieci samoorganizujące mają zwykle dosyć dużo wejść. Na rysunku 10.3 pokazano wprawdzie zaledwie trzy wejścia, ale to tylko dlatego, żeby nie komplikować rysunku – przy trzydziestu wejściach plątanina połączeń byłaby

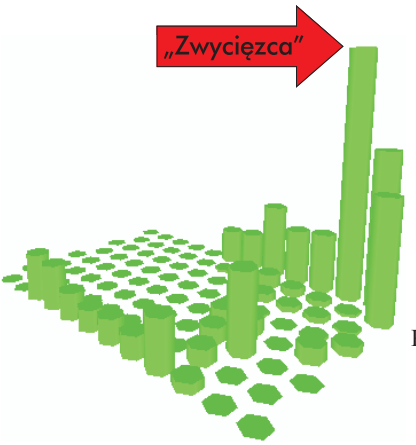



Rys.10.3. Struktura samoorganizującej się sieci neuronowej

niemożliwa do prześledzenia! Wykorzystywane w praktyce sieci tego typu mają typowo kilkanaście (a często także kilkadziesiąt) wejść, a ich użyteczność w kontekście różnych zastosowań jest z reguły tym większa, im więcej sygnałów wejściowych taka sieć potrafi obsłużyć. Jednak co ważniejsze i **trudniejsze**: sieci te mają zwykle jeszcze więcej wyjść, gdyż składają się z ogromnej liczby neuronów, tworzących łącznie tak zwaną warstwę topologiczną. Tu określenie „dużo neuronów" oznacza przynajmniej kilkadziesiąt, często kilkaset, nierzadko nawet kilka tysięcy neuronów – więc jest to naprawdę o wiele więcej niż we wszystkich sieciach, które widziałeś do tej pory. Jak zrozumieć, co właściwie sieć nam przedstawia w tej ogromnej warstwie topologicznej jako wynik swojej pracy?

Z pomocą przychodzi nam tu koncepcja, która już wcześniej wielokrotnie występowała w tej książce, mianowicie pomysł rywalizacji neuronów oraz wyłaniania zwycięzcy. Po podaniu sieci dowolnego konkretnego sygnału wejściowego wszystkie neurony warstwy topologicznej obliczają swoje sygnały wyjściowe, będące ich odpowiedziami na ten sygnał. Wśród tych sygnałów jeden z reguły jest największy – i neuron, który właśnie ten największy sygnał wyprodukował, staje się zwycięzcą (rys. 10.4).

Proces samoorganizacji, który będzie dokładnie opisany w dalszych podrozdziałach, skutkuje między innymi tym, że taki zwycięski neuron jest tylko jeden oraz powoduje, że przewaga „zwycięzcy" nad „konkurentami" jest bardzo duża i wyraźna. Jedynie w przypadku, gdy sieć (po procesie samoorganizacji) dostanie na wejście jakiś zupełnie nietypowy sygnał (niepodobny do żadnego sygnału, który występował w zbiorze uczącym) – są trudności z wyłonieniem zwycięzcy, bo wszystkie neurony warstwy topologicznej produkują bardzo słabe (i w gruncie rzeczy mało zróżnicowane) sygnały. Jest to jednak sytuacja nietypowa.




Rys. 10.4. Przykładowy rozkład wartości sygnałów wyjściowych neuronów warstwy topologicznej oraz wyłoniony w wyniku oceny tych sygnałów „zwycięzca"

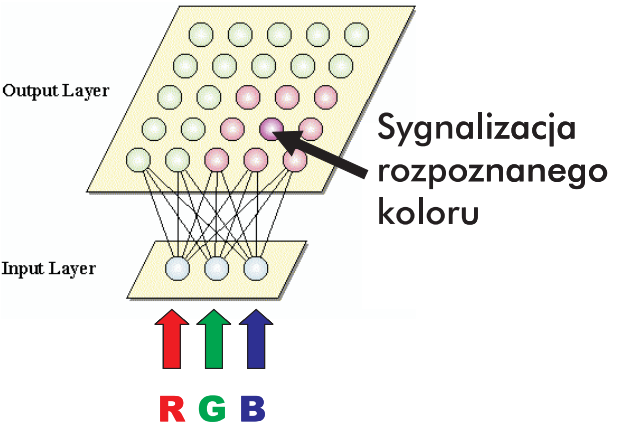
## 10.2. Na czym polega samoorganizacja w sieci i do czego może się przydać?

Nawet ta niewielka cząstka wiedzy, jaką zdobyłeś w poprzednim podrozdziale, pozwala Ci zorientować się, do czego zdolne są sieci, w których dochodzić będzie do samoorganizacji. Tworzą one – całkiem same, wyłącznie na podstawie obserwacji danych wejściowych – pewne odwzorowania zbioru sygnałów wejściowych w zbiór sygnałów wyjściowych, przy czym są to odwzorowania spełniające pewne ogólne kryteria (zaraz je omówię), w żaden jednak sposób nie determinowane z góry przez twórcę sieci ani przez jej użytkownika. Struktura i właściwości potrzebnego nam odwzorowania powinny powstać **same** w następstwie procesu skoordynowanego samouczenia wszystkich elementów sieci. Takie spontaniczne tworzenie przez sieć potrzebnego odwzorowania sygnałów nazywane jest właśnie **samoorganizacją.** Z pewnego punktu widzenia jest to jedynie jeszcze jedna forma samouczenia, jeśli jednak spojrzysz na jej skutki (a zobaczysz tych skutków wiele, bo program używany w tym rozdziale dostarczy Ci wielu niezwykle ciekawych obrazków), to zapewne sam chętnie przyznasz, że mamy tu do czynienia z wyraźnie **wyższym** stopniem adaptacji sieci, angażującym nie tylko optymalizację parametrów każdego neuronu z osobna, ale – co jest właśnie nowością – koordynację działań neuronów, wprowadzającą do obliczeń bardzo pożądane efekty **grupowania** i **kolektywności.**

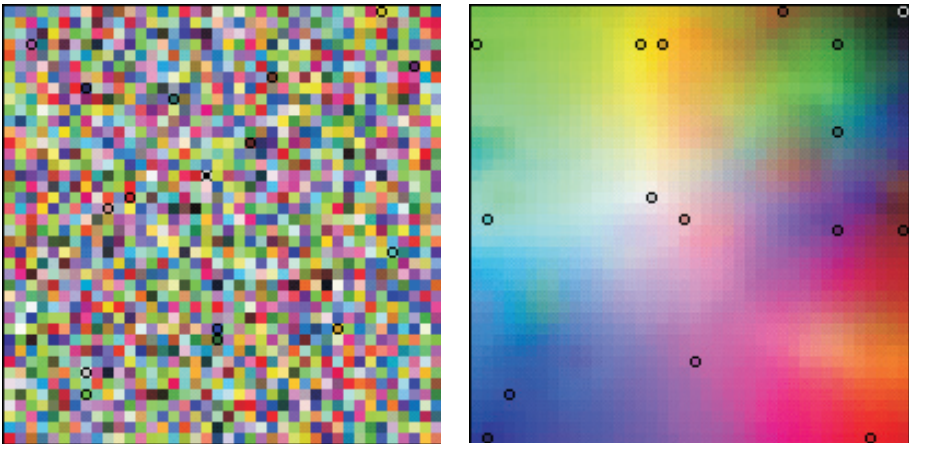
Wyjaśnijmy te pojęcia. Efekt **grupowania** polega na tym, że sieć w procesie samoorganizacji usiłuje rozdzielać dane wejściowe, wyróżniając wśród nich pewne **klasy podobieństwa**. Działa to tak, że wśród obiektów wejściowych (opisywanych przez wejściowe sygnały) wykrywane są (całkiem automatycznie!) takie ich grupy, w których można umieścić sygnały podobne do siebie nawzajem i które jednocześnie są wyraźnie odmienne od sygnałów przypisanych do innych grup.

Takie grupowanie danych jest bardzo przydatne w wielu zastosowaniach, wypracowano więc szereg specjalizowanych technik matematycznych pozwalających na analizowanie danych (najczęściej statystyczne) i na tworzenie właśnie takich grup danych. Wzmiankowane techniki nazywane są po polsku „analizą skupień", a w literaturze światowej są szerzej znane pod angielską nazwą *cluster analysis.* Są one dość chętnie stosowane w ekonomii – na przykład do wykrywania, które przedsiębiorstwa są podobne do siebie i mogą w związku z tym stwarzać podobne rokowania rentowności inwestycji, albo w medycynie – do badania, które objawy wskazują na różne odmiany tej same choroby, a które wskazują już na obecność nowej, być może nie znanej jednostki chorobowej.

Na rysunku 10.5 masz przedstawiony przykład sieci neuronowej, która potrafi grupować wejściowe sygnały (piksele kolorowego obrazka, kodowane w typowy sposób za pomocą trzech składowych RGB) według kryteriów podobieństwa ich kolorów. Na rysunku 10.6 pokazano efekt działania tej sieci bez procesu samoorganizacji (po lewej stronie) oraz po procesie samoorganizacji (po prawej stronie). Rysunek 10.6 powstał w ten sposób, że w każdym miejscu, w którym znajduje się neuron warstwy topologicznej, narysowano kwadracik wypełniony takim kolorem wejściowego piksela, który powoduje, że ten właśnie neuron staje się „zwycięzcą".



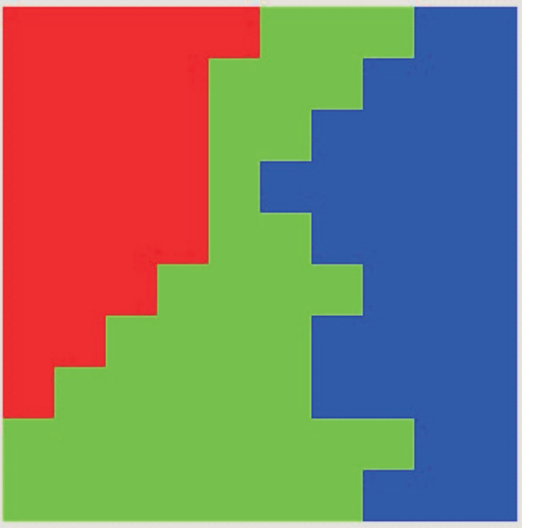

Rys. 10.5 Sieć grupująca wejściowe sygnały (składowe odpowiadające typowemu kodowaniu cyfrowego obrazu) w kolekcje odpowiadające poszczególnym możliwym barwom



Rys. 10.6. Obraz działania sieci z rysunku 10.5. Po lewej stronie – przed procesem samoorganizacji, po prawej – po procesie samoorganizacji

Grupowanie razem pikseli obrazka, mających podobny kolor, nie jest zadaniem, które by dawało jakieś szczególne korzyści, więc rysunek 10.6 możesz obejrzeć jako ciekawostkę, a potem możesz wzruszyć ramionami i powiedzieć: „No dobrze, ale po co to komu?" No to popatrz na rysunek 10.7. Na tym rysunku pokazano, co powstało w warstwie topologicznej sieci samoorganizującej się, której kazano porównać dane dotyczące różnych przedsiębiorstw. Na wejściu sieci podawano różne dane: rodzaj prowadzonej działalności gospodarczej, posiadany kapitał, liczbę zatrudnionych pracowników, dane bilansowe za kilka ostatnich kwartałów, obrazujące uzyskane zyski lub poniesione straty itp. Sieć pogrupowała te przedsiębiorstwa i tak ustawiła swoje parametry, że każdy jej neuron miał zwyczaj stawać się „zwycięzcą", gdy pokazywano informacje o „jego" przedsiębiorstwie. Potem odczekano rok i sprawdzono, co się stało z tymi przedsiębiorstwami. Okazało się, że część z nich upadła albo są one zagrożone bankructwem – to te, których neurony zaznaczono na rysunku 10.7 kolorem czerwonym. Inne przeciwnie, okazały się dotknięte gospodarczą stagnacją. Neurony, które zostały do nich przypisane w **automatycznym** procesie samoorganizacji, zaznaczone są na rys. 10.7 kolorem błękitnym. No i wreszcie była grupa przedsiębiorstw, które rozwijały się harmonijnie – domyślasz się, że to te zielone.

Takich bardzo efektownych przykładów samoorganizacji, prowadzących do uporządkowania pewnych danych, można znaleźć mnóstwo w Internecie – wystarczy w przeglądarce Google wpisać hasło „sieci samoorganizujące się" albo lepiej – powszechnie używany skrót SOM (*Self-Organizing Maps*). Robiąc to bardzo szybko, przekonasz się, że opisywane tu zadania grupowa-



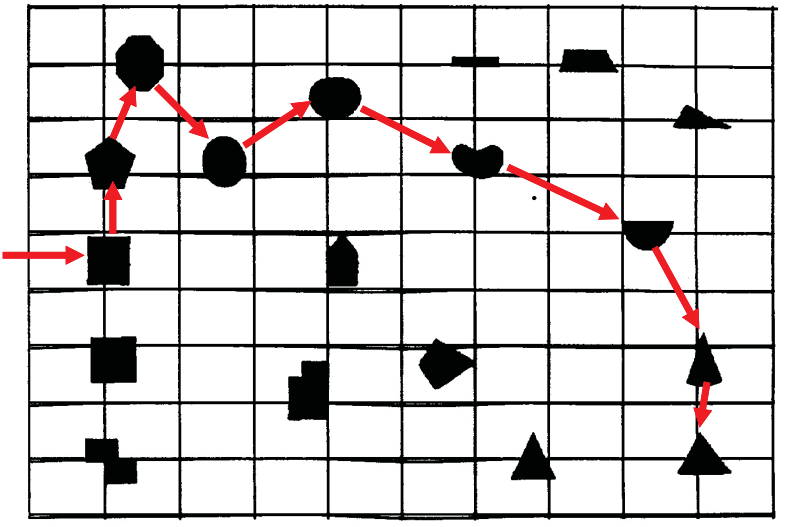
Rys. 10.7. Efekt grupowania przedsiębiorstw w samoorganizującej się sieci neuronowej. Opis w tekście

nia wejściowych danych w klasy podobieństwa są potrzebne i znajdują praktyczne zastosowania.

Sieci neuronowe dokonujące procesu samoorganizacji są więc bardzo atrakcyjnym narzędziem do realizacji zadań grupowania wejściowych danych i tworzenia wśród nich klas podobieństwa. Atrakcyjność neuronowego podejścia do rozważanego problemu polega głównie na tym, że proces samoorganizacji może być tu prowadzony absolutnie automatycznie i spontanicznie, a twórca sieci nie musi jej przy tym udzielać żadnych wskazówek, ponieważ komplet potrzebnych informacji zawarty jest w samych wejściowych danych (z których sieć sama wydobędzie obserwację, że jedne są podobne do siebie nawzajem, a inne nie). Ponadto po nauczeniu sieci grupowania wejściowych danych powstaną bardzo użyteczne systemy – „zwycięskie" neurony, które wyspecjalizowały się w rozpoznawaniu poszczególnych klas sygnałów wejściowych, są więc ich detektorami i mogą być użyte do ich sygnalizowania.

Omówiwszy w ten sposób krótko, na czym polega efekt grupowania w samoorganizujących się sieciach neuronowych, powiem Ci jeszcze w kilku słowach, co miałem na myśli mówiąc o **kolektywności** działania sieci.

Otóż sieci, w których zachodzi samoorganizacja, są tak zorganizowane, że to, co rozpoznaje jeden neuron, w dużej mierze zależy także od tego, co rozpoznają inne neurony, znajdujące się w jego pobliżu. W ten sposób **zbiorowość** neuronów (czyli ich kolektyw) może w sposób pełniejszy i bogatszy przetwarzać informacje niż każdy z neuronów z osobna wzięty. Obejrzyj rysunek 10.8.



Rys. 10.8. Efekt rozpoznawania figur geometrycznych w samoorganizującej się sieci neuronowej. Opis w tekście

Na rysunku tym pokazano wynik moich badań związanych z rozpoznawaniem przez samoorganizującą się sieć neuronową prostych figur geometrycznych, pokazywanych jej jako obrazy. W miejscach, gdzie sieć ustawiła (całkiem sama!) neurony sygnalizujące poszczególne kształty pokazywanych jej figur geometrycznych – narysowano odpowiednie figury. Odbądź wraz ze mną krótką podróż po tej mapie, a przekonasz się, że to rozmieszczenie neuronów na pewno nie jest przypadkowe!

Zaczniemy – zgodnie z czerwonymi strzałkami, które dorysowałem na rysunku – od neuronu, który staje się „zwycięzcą", ilekroć sieci pokazany zostanie **kwadrat**. Niedaleko od niego jest neuron sygnalizujący **pięciokąt**, a nieco dalej – neuron sygnalizujący **ośmiokąt**. Słusznie: pięciokąt jest bardziej podobny do kwadratu niż ośmiokąt! Nieopodal ośmiokąta ulokował się neuron rozpoznający **koło.** Od niego krótka droga wiedzie do neuronu sygnalizującego **elipsę**, która z kolei jest niedaleko od neuronu rozpoznającego **półkole. Półkole** ma wyraźnie powinowactwo do mniejszego kawałka koła, nazywanego przez nas **sektorem,** a ten z kolei ma wyraźnie zbliżony kształt do **trójkąta**…

Jak widzisz, efekt kolektywnego działania sieci samoorganizujących się jest bardzo ciekawy. Co więcej, jest to efekt, będący przyczynkiem do ogólniejszych rozważań: Otóż system, czyli właśnie zbiorowość odpowiednio powiązanych i współpracujących elementów, stwarza możliwość uzyskania nowych form zachowania i nowych postaci działań, znacznie bogatszych, niż by można było oczekiwać biorąc pod uwagę każdy z elementów z osobna. Na przykład każdy pojedynczy owad jest dość głupim i prymitywnym zwierzęciem, a tymczasem zbiorowości owadów (rodzina pszczela, mrowisko, kopiec termitów itp.) zdolne są do celowych, skomplikowanych i bez wątpienia inteligentnych działań. Kiedyś opiszę moje doświadczenia, jakie sam zebrałem podczas komputerowego modelowania rodziny pszczelej i wtedy napiszę o tym więcej – ale już nie w tej książce.

Wracając do sieci z samoorganizacją, trzeba stwierdzić, że są one dość wygodnym, użytecznym i chętnie stosowanym narzędziem, a ponadto – a może nawet przede wszystkim – są bardzo interesującym obiektem badań. Szczegóły procesu samoorganizacji zachodzącego w sieciach neuronowych poznasz dokładniej studiując wiadomości przedstawione w tym rozdziale i – jak zwykle – wykonując doświadczenia ze specjalnie przygotowanym w tym celu programem. Kluczem do zrozumienia i używania sieci samoorganizujących się jest pojęcie **sąsiedztwa** neuronów – i nim się właśnie teraz przez chwilę zajmiemy.

## 10.3. Jak wprowadza się do sieci sąsiedztwo?

Żeby dobrze zrozumieć budowę (i działanie!) sieci samoorganizującej się, spróbujmy najpierw porównać jej działanie, w skrócie opisane powyżej, z tym, co potrafiły zrobić proste sieci samouczące się, opisane w poprzednim rozdziale. Otóż w wyniku przemożnego wpływu losowych wartości początkowych na przebieg prostego procesu samouczenia, opisanego w rozdziale 9, twórca zwyczajnej sieci samouczącej się nie ma żadnego wpływu na to, które neurony czego się nauczą. Czasem rozłożenie neuronów sygnalizujących (po zakończonym procesie samouczenia) określone zdarzenia lub określone zjawiska bywa bardzo niewygodne i niekorzystne, nie możesz go jednak zmienić bez „ręcznej" ingerencji w zachowanie sieci, co w ogólnym przypadku jest dość trudne do wykonania, a ponadto jest sprzeczne z przyjętą zasadą samouczenia. Z kolei w wyniku wprowadzenia do sieci samej tylko **konkurencji** może się zdarzyć, że tylko niektóre (zwykle dość nieliczne) neurony o dużych „wrodzonych uzdolnieniach" podlegać będą procesowi uczenia, pozostałe natomiast pozostaną poza procesem uczenia i będą – z punktu widzenia celów, dla których sieć zorganizowano – właściwie stracone. Mogłeś to zaobserwować na przykład w programie **Example 10c**, który podawał – między innymi – informacje na temat tego, które neurony (konkretne numery) stawały się „zwycięzcami" dla poszczególnych klas i w dalszym ciągu stawały się detektorami wykrywającymi obiekty danej klasy. Numery te pojawiały się zdecydowanie w sposób chaotyczny, bez żadnego związku z lokalizacją wykrywanych obiektów. Tymczasem przypisanie funkcji wykrywania określonych obiektów wejściowych do pewnego konkretnego, właśnie tego, a nie innego neuronu – może stanowić ważny czynnik zwiększający lub zmniejszający użyteczność sieci. Jeśli – na przykład – niektóre sygnały wejściowe są do siebie jakoś podobne (w dowolnym sensie), to może Ci zależeć, żeby ich pojawianie się było sygnalizowane przez neurony sąsiadujące ze sobą w sieci. Tak działała sieć samoorganizująca się, której wynik pracy pokazałem Ci na rysunku 10.8. Tymczasem w „czystej" sieci samouczącej się nie masz na to żadnego wpływu i musisz pokornie przyjąć to, co się zdarzy.

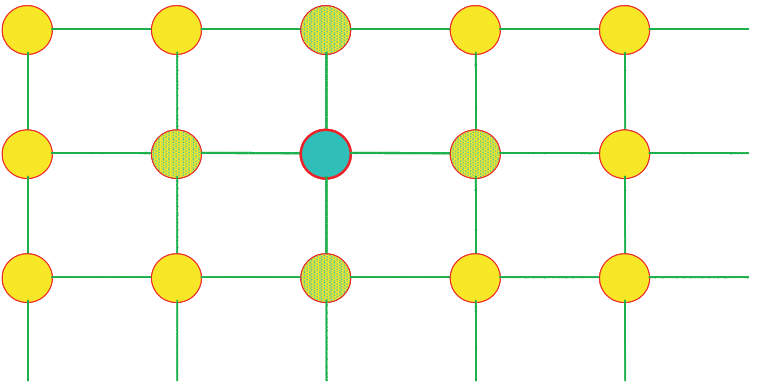
Radą na te wszystkie bolączki jest wprowadzenie do samouczącej się sieci jeszcze jednego mechanizmu: **sąsiedztwa**. Po raz pierwszy sąsiedztwo neuronów w sieci wykorzystał (w latach 70.) fiński badacz Teuvo Kohonen i dlatego w literaturze sieci wykorzystujące sąsiedztwo (i zwykle także angażujące konkurencję między neuronami) nazywane są *sieciami Kohonena.*

Pokażę Ci teraz, na czym owo sąsiedztwo polega, a potem opowiem Ci, co ono daje.

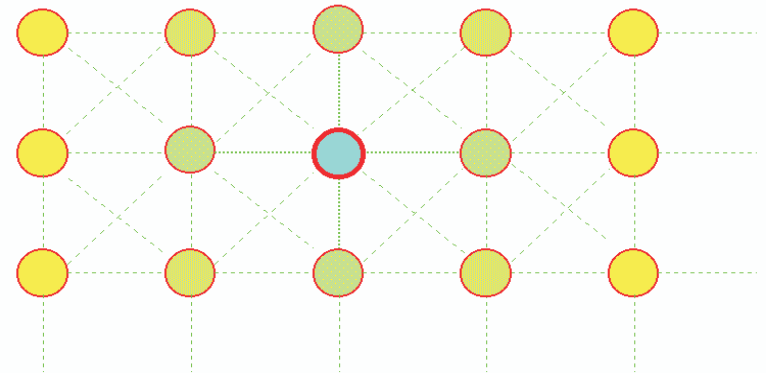
Do tej pory neurony sieci rozważałeś jako jednostki w znacznym stopniu niezależne od siebie. Były one wprawdzie łączone ze sobą i przekazywały

sobie nawzajem sygnały (jak to w sieci) – ale ich wzajemne **położenie** w warstwach nie miało żadnego znaczenia. Co najwyżej dla celów porządkowych wprowadzana była (wykorzystywana także do organizacji obliczeń w programach symulujących sieć) numeracja neuronów – i to wszystko. Tymczasem w rozważanej tutaj sieci samoorganizującej się bardzo ważne okazuje się to, które neurony warstwy topologicznej uznasz za **sąsiadujące** ze sobą, gdyż będzie to w zasadniczy sposób rzutowało na zachowanie całej sieci.

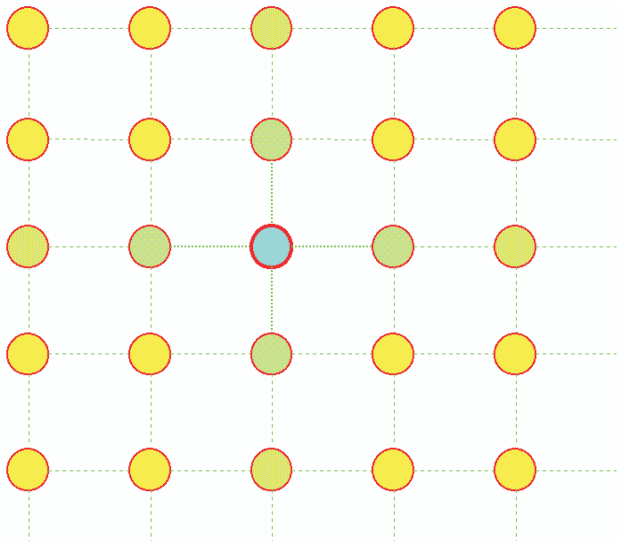
Najczęściej neurony warstwy topologicznej kojarzymy z punktami jakiejś mapy, rysowanej albo wyświetlanej na ekranie komputera, dlatego typowo rozpatrujemy **sąsiedztwo dwuwymiarowe:** neurony rozważa się tak, jakby były rozmieszczone w węzłach regularnej siatki złożonej z wierszy i kolumn. W siatce takiej każdy neuron ma przynajmniej czterech sąsiadów: dwóch w poziomie (z lewej i z prawej) i dwóch w pionie (u góry i u dołu) – rys. 10.9.



Rys. 10.9. Neuron i jego sąsiedzi w sieci Kohonena

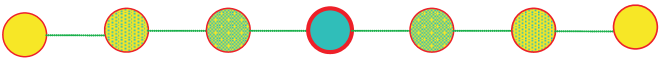


Rys. 10.10. Rozbudowane sąsiedztwo w sieci Kohonena



Rys. 10.11. Rozszerzony zakres sąsiedztwa w sieci Kohonena

Jeżeli potrzeba – można sąsiedztwo traktować szerzej: dopuścić także sąsiadów po przekątnej (rys. 10.10) albo dołączyć do sąsiedztwa neurony znajdujące się w dalszych rzędach lub kolumnach (rys. 10.11). To, jak zdefiniujesz sąsiedztwo, zależy wyłącznie od Ciebie – możesz na przykład tak opisać sieć, że **sąsiedztwo będzie jednowymiarowe** (neurony będą wtedy tworzyły długi łańcuch i każdy neuron będzie miał sąsiadów poprzedzających go w łańcuchu i sąsiadów, którzy następują po nim – rys. 10.12).



Rys. 10.12. Specjalny przypadek: sąsiedztwo jednowymiarowe

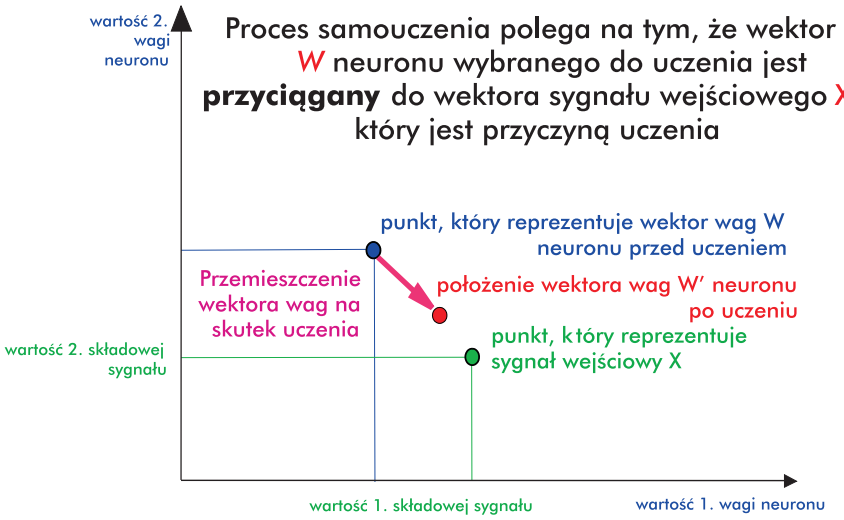
W specjalistycznych zastosowaniach sieci samoorganizujących się można wprowadzić sąsiedztwo trójwymiarowe (neurony ze swoimi sąsiadami wyglądają wtedy jak atomy tkwiące w sieci krystalicznej – na pewno widziałeś już takie obrazki, a mnie się nie chce wysilać i czegoś podobnego teraz tu rysować, więc sam to sobie wyobraź), a bywa także rozważane sąsiedztwo cztero-, pięcio- i więcej wymiarowe (zgadnij, dlaczego Ci tego nie narysuję?). Jednak zdecydowanie najwięcej praktycznych zastosowań znajdują sieci jedno i dwuwymiarowe, dlatego do nich ograniczymy dalej wszystkie rozważania.

**Sąsiedztwo** dotyczy oczywiście wszystkich neuronów sieci: każdy neuron posiada komplet sąsiadów i sam z kolei jest sąsiadem dla innych neuronów.

Tylko neurony znajdujące się na brzegu sieci nie mają kompletu sąsiadów, ale można temu czasem zaradzić wprowadzając specjalną umowę (na przykład „domykając" sieć w taki sposób, że neurony z górnego brzegu są traktowane jako sąsiednie w stosunku do neuronów z dolnego brzegu; podobnie domyka się lewą i prawą krawędź sieci).

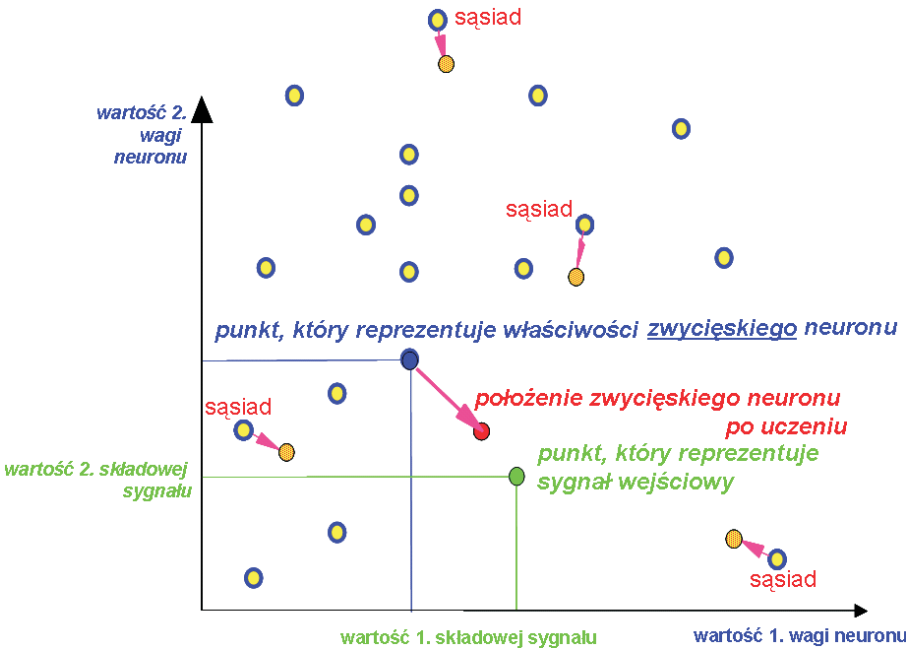
## 10.4. Co wynika z tego, że jakieś neurony uważamy za sąsiednie?

Fakt, że pewne neurony uznane są za sąsiednie, ma bardzo duże znaczenie. Kiedy w procesie uczenia jakiś neuron zostaje zwycięzcą i podlega procesowi uczenia – wraz z nim uczy się także jego sąsiadów. Zaraz Ci pokażę, jak to się dzieje, ale wcześniej przypomnę Ci, jak w sieciach samouczących się przebiega uczenie pojedynczych neuronów (rys. 10.13).




Rys. 10.13. Proces samouczenia pojedynczego neuronu

A teraz porównaj rysunek 10.14. Zauważ, co z niego wynika: Neuron zwycięzca (oznaczony granatowym punktem) podlega uczeniu, ponieważ **jego** pierwotne współczynniki wagowe były podobne do składowych sygnału pokazanego w trakcie procesu uczenia (zielony punkt). Zatem w tym miejscu następuje jedynie wzmocnienie i skonkretyzowanie naturalnych, „wrodzonych" preferencji tego neuronu, co widziałeś już wcześniej w innych sieciach samouczących się. Na rysunku wygląda to tak, jak gdyby „zwycięzca" był



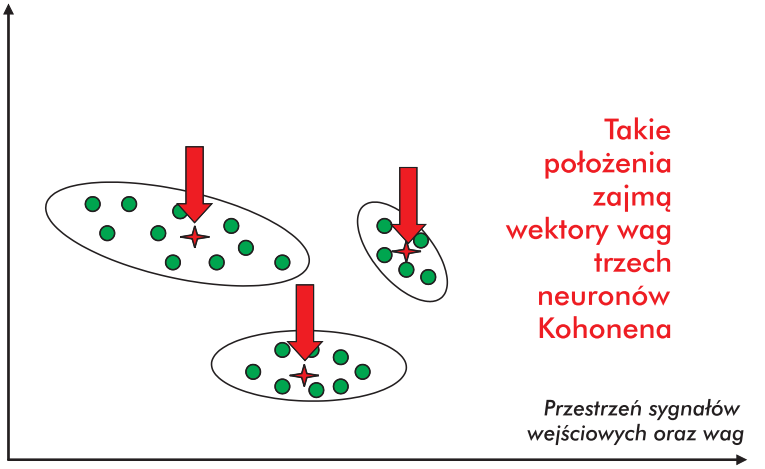

Rys. 10.14. Widok procesu uczenia rozszerzonego na sąsiadów „zwycięzcy"

silnie przyciągany przez wejściowy punkt, który spowodował, że to on właśnie został zwycięzcą – jego wektor wag (i punkt, reprezentujący ten wektor na rysunku) silnie przemieszcza się i zbliża do punktu reprezentującego sygnał wejściowy. Sąsiedzi zwycięskiego neuronu (żółte punkty lekko podbarwione na czerwono) nie mają tego szczęścia – niezależnie jednak od tego, jakie były ich początkowe wagi i wynikające z nich sygnały wyjściowe, **są** oni uczeni tak, żeby w przyszłości mieli skłonność do rozpoznawania tego właśnie sygnału wejściowego, dla którego „wybitnie uzdolniony" sąsiad okazał się zwycięzcą! Żeby jednak było sprawiedliwie – sąsiedzi są uczeni nieco mniej intensywnie niż zwycięzca (strzałki, wskazujące na wielkości ich przemieszczeń, są wyraźnie krótsze). Jednym z ważnych parametrów definiujących właściwości sieci z sąsiedztwem jest właśnie współczynnik określający, na ile słabiej należy uczyć sąsiadów niż samego zwycięzcę. Zauważ, że inne neurony (żółte punkty), których parametry niejednokrotnie znacznie bardziej predestynowały je do uczenia (były one znacznie bliżej wejściowego punktu) – nie podlegały w tym kroku żadnemu uczeniu.

Jaki będzie skutek takiego dziwnego sposobu uczenia?

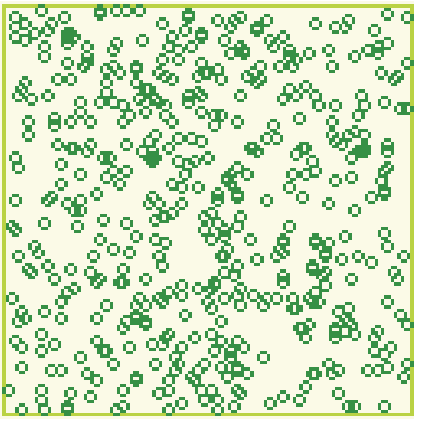
Otóż jeśli sygnały wejściowe do sieci będą nadchodziły w taki sposób, że będą wyraźnie wyróżnione ich skupiska, to poszczególne neurony sieci

będą usiłowały zająć (poprzez swoje wektory wag) pozycje w centrach tych skupisk, zaś sąsiednie neurony będą „obstawiały" sąsiednie skupiska. Taką sytuację przedstawia rysunek 10.15, na którym zielonymi kółkami oznaczono punkty reprezentujące sygnały wejściowe, natomiast czerwone gwiazdki odpowiadają położeniu (w tym samym układzie współrzędnych) wektorów wag poszczególnych neuronów.



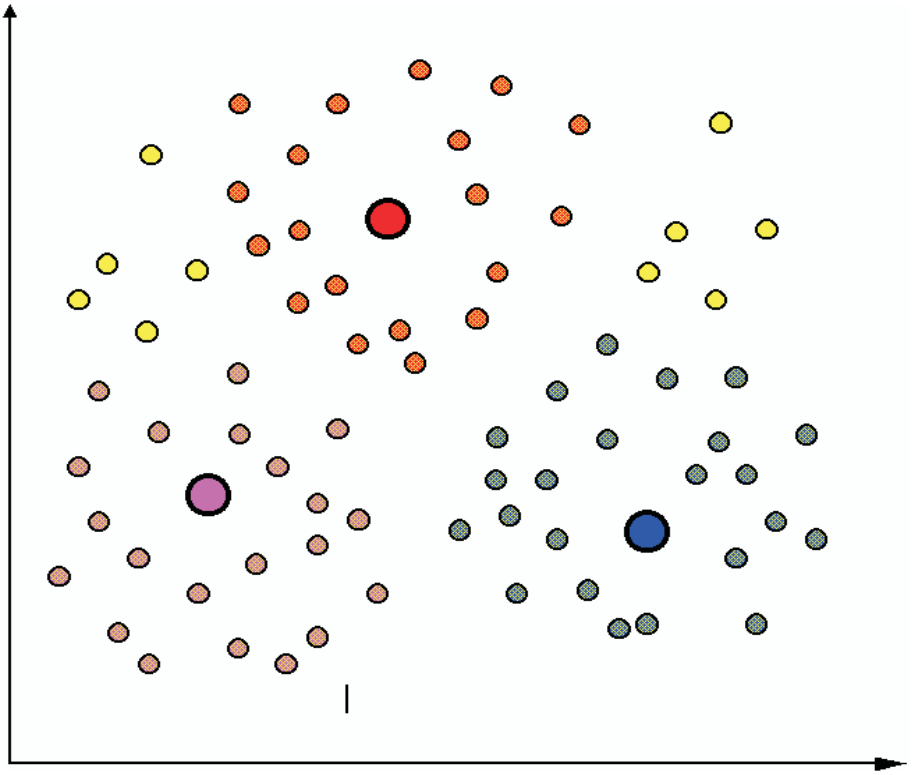

Rys. 10.15. Efekt uczenia sieci Kohonena, gdy sygnały wejściowe tworzą wyraźne skupiska

Znacznie gorsza sytuacja pojawi się wtedy, gdy sygnały wejściowe będą równomiernie rozłożone w pewnym obszarze przestrzeni sygnałów wejściowych, tak jak to przedstawiono na rysunku 10.16. Wówczas neurony sieci będą miały tendencje do tego, by „podzielić" się funkcją rozpoznawania tych



Rys. 10.16. Przypadek uczenia sieci zbiorem danych w przybliżeniu równomiernie rozsianych wewnątrz pewnego obszaru przestrzeni sygnałów wejściowych

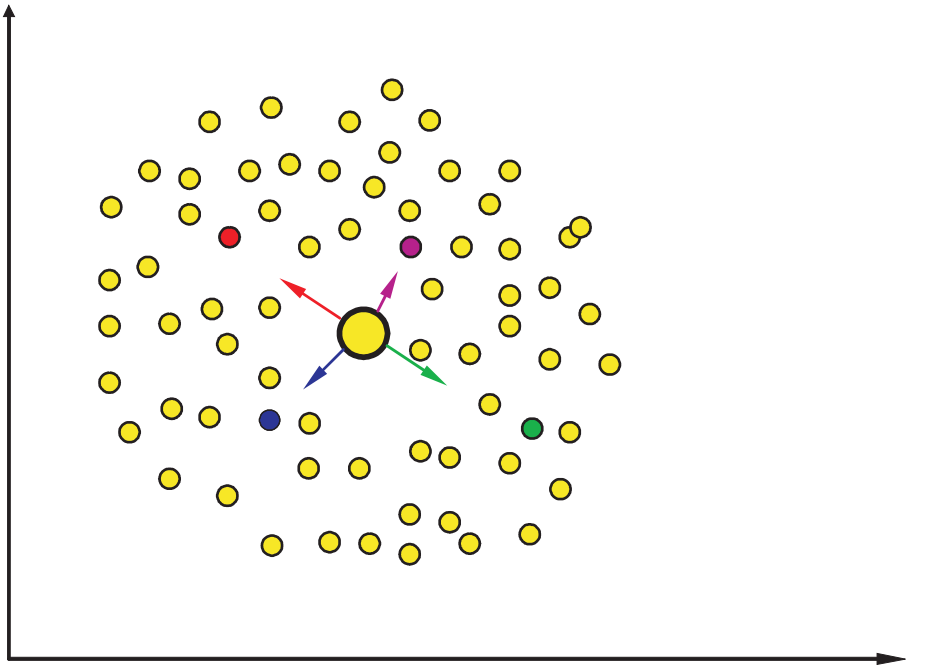
sygnałów w taki sposób, by każdy podzbiór sygnałów wejściowych miał swojego „anioła stróża” w postaci neuronu, który będzie wykrywał i rozpoznawał wszystkie sygnały z jednego podobszaru, inny będzie wykrywał sygnały z innego podobszaru itd. Ilustruje to rysunek 10.17.



Rys. 10.17. Lokalizacja wektorów wag neuronów (duże kółka) w miejscach, w których mogą one reprezentować pewne podgrupy (zbiory jednakowo pokolorowanych małych kółek) danych wejściowych

Przy oglądaniu tego rysunku konieczny jest – jak się wydaje – jeden komentarz. Otóż nie od razu może być dla Ciebie oczywiste, że w przypadku **losowego** pojawiania się zbioru punktów z pewnego obszaru i **systematycznego** prowadzenia nauki – położenie, jakie zajmie punkt reprezentujący wagi neuronu będzie położeniem centralnym w tym zbiorze. Jednak tak właśnie jest, co można prześledzić na rysunku 10.18.

Jak widać z tego rysunku, gdy neuron (reprezentowany oczywiście jak zwykle przez swój wektor wag) zajmie już położenie w centrum „mgławicy” punktów, które ma rozpoznawać, wówczas dalszy tok nauki nie jest w stanie

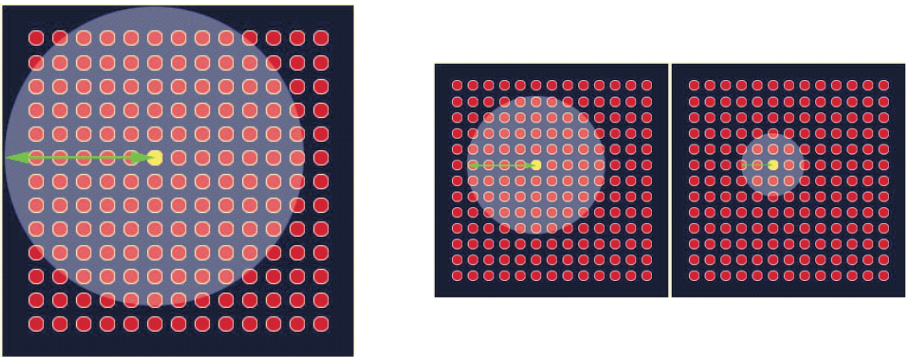


Rys. 10.18. Wzajemna kompensacja wpływów różnych punktów podczas nauczania neuronu, gdy już zajmie on centralne położenie

na trwale przesunąć go z tego położenia, ponieważ różne punkty pojawiające się w ciągu uczącym wywołują przesunięcia, które się wzajemnie kompensują. Aby ograniczyć zakres tego „myszkowania" neuronu wokół jego końcowego położenia w sieciach Kohonena, stosuje się na ogół regułę **uczenia z malejącym współczynnikiem uczenia**, w związku z tym zasadnicze ruchy związane z odnalezieniem przez każdy neuron właściwej dla niego lokalizacji odbywają się głównie na początku uczenia (kiedy współczynnik uczenia jest jeszcze duży). Natomiast punkty pokazywane pod koniec procesu uczenia bardzo słabo już wpływają na położenie neuronu, który po pewnym czasie ustala swoje położenie i nie zmienia go więcej.

Obok tego procesu osłabiania kolejnych korekt w trakcie uczenia sieci zachodzi także inny proces: zakres sąsiedztwa systematycznie maleje. Na początku zmiany wynikające z sąsiedztwa dotyczą przy każdym kroku uczenia wielu neuronów, stopniowo sąsiedztwo to jest zawężane i zacieśniane, a na końcu każdy z neuronów jest samotny i pozbawiony sąsiadów (rys. 10.19).

Jeśli się zastanowisz nad wszystkimi wyżej podanymi wiadomościami, to zauważysz, że po skończonej nauce neurony warstwy topologicznej po-

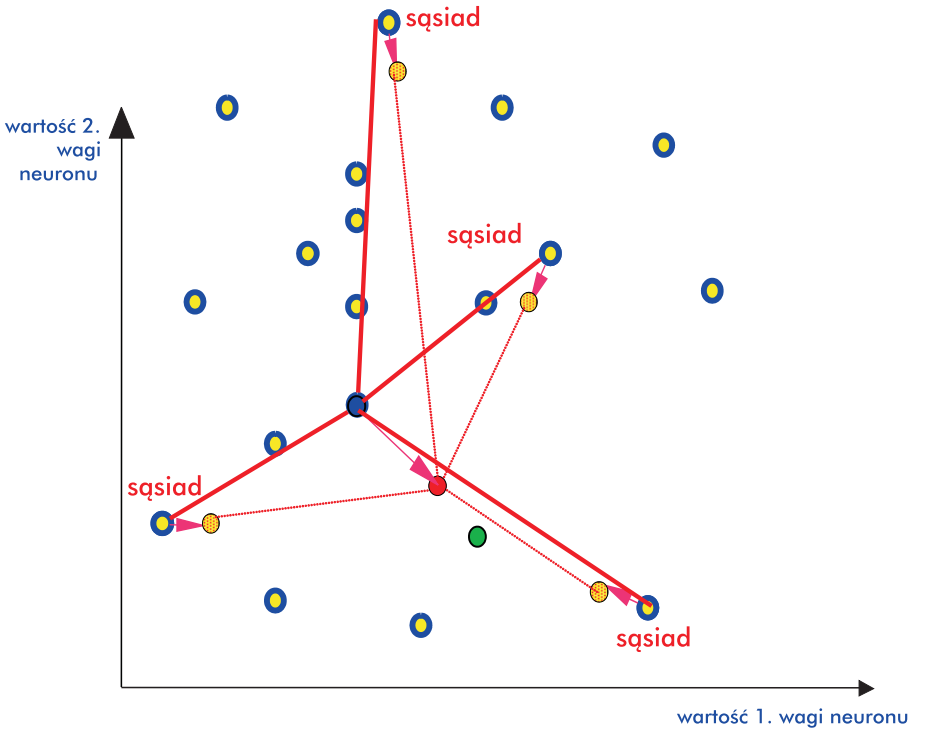


Rys. 10.19. Malenie zasięgu sąsiedztwa w trakcie trwania uczenia sieci Kohonena

dzielą pomiędzy siebie przestrzeń sygnałów wejściowych w taki sposób, że każdy obszar tej przestrzeni będzie sygnalizowany przez osobny neuron. Co więcej, w następstwie działania sąsiedztwa, te neurony, które Ty uznałeś za sąsiednie – będą wykazywały skłonność do rozpoznawania bliskich – czyli podobnych do siebie obiektów wejściowych. Okaże się to własnością bardzo wygodną i bardzo przydatną, gdyż tego typu samoorganizacja jest kluczem do szczególnie inteligentnych zastosowań sieci jako samoorganizujących się odwzorowań. Rozważaliśmy to na konkretnych przykładach w pierwszych podrozdziałach tego rozdziału.

Przy prezentacji wyników uczenia sieci Kohonena napotkasz jeszcze jedną trudność, którą warto omówić, zanim jeszcze zetkniesz się z konkretnymi wynikami symulacji, żeby potem wszystko było już zupełnie jasne. Otóż przedstawiając wyniki (w postaci zachodzących w trakcie uczenia zmian lokalizacji punktów odpowiadających poszczególnym neuronom) musisz mieć możliwość śledzenia także tego, co się dzieje z **sąsiednimi** neuronami. Na rysunku 10.14 mogłeś łatwo skorelować to, co się działo ze „zwycięskim" neuronem oraz z jego sąsiadami, ponieważ punktów było niewiele i wykrycie sąsiadów na podstawie zmienionego koloru było łatwe i wygodne. Podczas symulacji będziesz jednak niekiedy miał do czynienia z setkami neuronów i taka technika prezentacji jest niemożliwa do utrzymania. Dlatego podczas prezentacji działania sieci Kohonena nagminnie stosuje się sposób rysowania „mapy" położeń neuronów z zaznaczonymi relacjami sąsiedztwa – jak na rysunku 10.20.

Na rysunku możesz zauważyć, że punkty odpowiadające sąsiednim neuronom przedstawiane są jako połączone liniami. Gdy w wyniku procesu uczenia przemieszczają się punkty – przemieszczają się także odpowiednie linie. Oczywiście, powinno to dotyczyć wszystkich neuronów i wszystkich relacji sąsiedztwa, ale na rysunku 10.20. dla maksymalnej jego czytelności pokaza-




Rys. 10. 20. Przebieg jednego kroku uczenia w sieci Kohonena z uwidocznieniem (za pomocą czerwonych linii), które neurony są sąsiadami

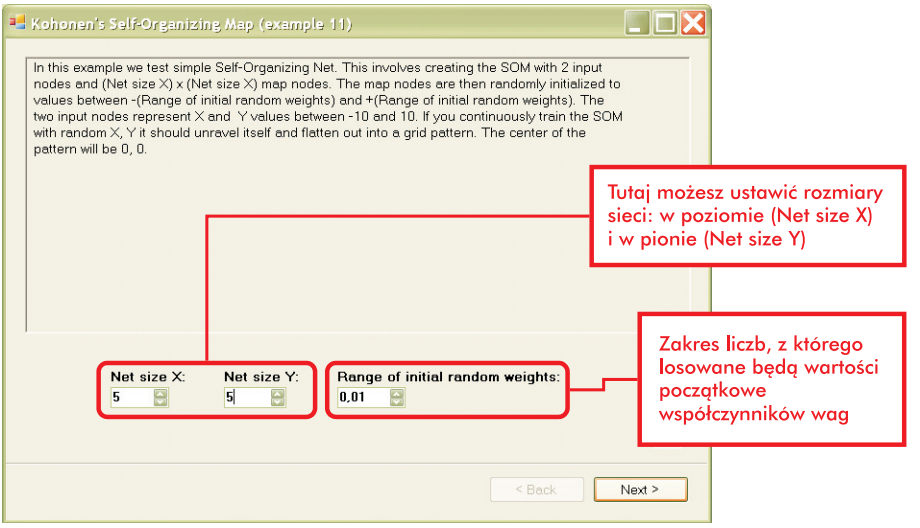
łem tylko te linie, które dotyczyły aktualnie „zwycięskiego" neuronu i jego sąsiadów, pominąłem natomiast wszystkie inne połączenia. W szczegółach dla całej sieci zobaczysz to za chwilę na przykładzie programu **Example 11**, który dla Ciebie przygotowałem.

## 10.5. Co potrafią zrobić sieci Kohonena?

Program **Example 11** ilustruje działanie sieci Kohonena. Żebyś jednak wiedział, jak z niego korzystać – najpierw kilka uwag ogólnych.

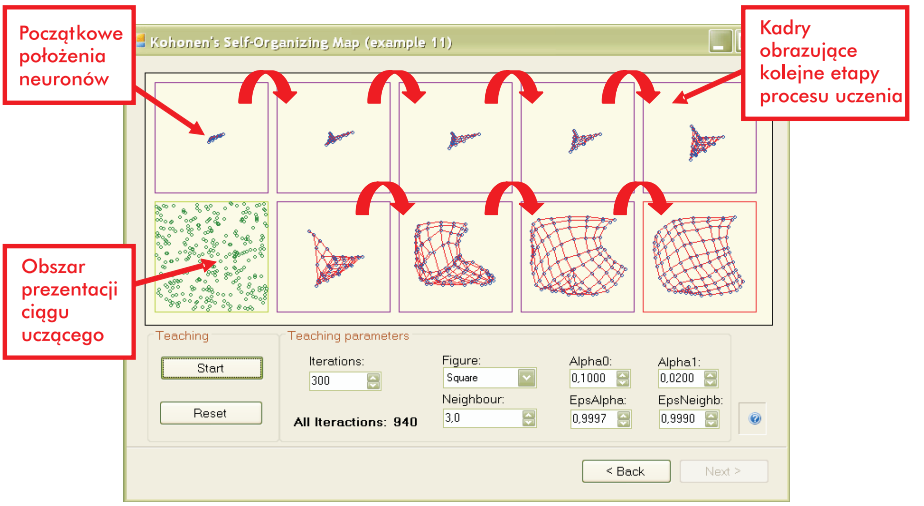
Zaraz po jego uruchomieniu pojawi się okno, gdzie możesz ustawić rozmiary sieci: w poziomie (**Net size X**) i w pionie (**Net size Y**) – rys. 10.21. Rozmiar sieci, jaki wybierzesz, zależy naturalnie tylko od Ciebie, jednak aby uzyskać bardziej przejrzyste wyniki, zacznij zabawę od sieci niezbyt dużych – na przykład dobrym pomysłem jest wybranie sieci o rozmiarach **5** × **5** neuronów. Taka sieć jest dość prymitywna i niewiele będzie z niej pożytku przy bardziej złożonych zadaniach, ale za to będzie się uczyła na tyle szybko, że

już po chwili zobaczysz, o co w tym wszystkim chodzi. Potem przyjdzie czas na sieci o większych rozmiarach. W samym kodzie programu nie ma jakichś szczególnych ograniczeń co wielkości sieci, niemniej ograniczono zakres wprowadzania tych parametrów od **1** do **100.** Przy uczeniu dużych sieci na mniej wydajnym sprzęcie musisz czasami wykazać cierpliwość, wiesz już bowiem z wcześniejszych doświadczeń, że dla ostatecznego nauczenia większej sieci **konieczne jest wykonanie od kilkuset do kilkunastu tysięcy kroków.** Oprócz ustawienia obydwu rozmiarów sieci możesz również w początkowym oknie (rys. 10.21) określić zakres liczb, z którego losowane będą wartości początkowe współczynników wag (**Range of initial random weights**). Na początek proponuję, żebyś zaakceptował ustawioną wartość domyślną, w dalszych doświadczeniach możesz jednak sobie bezpiecznie i ciekawie poeksperymentować z innymi (większymi) liczbami, bo to pokaże Ci, jak na działanie sieci wpływają „wrodzone" właściwości budujących ją neuronów.




Rys. 10.21. Początkowy dialog z programem **Example 11**

Po ustaleniu rozmiarów sieci i sposobu podania jej wartości początkowych przyciskiem **Next** przejdziesz do kolejnego okna (rys. 10.22), w którym będziesz śledził postępy procesu uczenia. W oknie tym położenia poszczególnych neuronów w przestrzeni sygnałów wejściowych zaznaczane będą niebieskimi okręgami, a czerwone linie biegnące od jednego okręgu do drugiego sygnalizować będą fakt, że te właśnie neurony są „sąsiadami" (zgodnie z przyjętymi zasadami ich wiązania ze sobą – rys. 10.20). Tam, gdzie w prze-




Rys. 10.22. Struktura ekranu programu **Example 11**

strzeni wejść zlokalizowany jest taki okrąg – tam mamy neuron rozpoznający i sygnalizujący pojawienie się sygnału z tego właśnie punktu (i jego bliskiego otoczenia, bo sieci neuronowe zawsze mają skłonności do pewnego uogólniania zdobytej wiedzy). W doświadczeniach z sieciami Kohonena zwykle pokazuje się losowo punkty z pewnego podobszaru przestrzeni sygnałów wejściowych. W rezultacie niebieskie kwadraty będą systematycznie rozchodziły się i rozprzestrzeniały po całej przestrzeni sygnałów wejściowych – a dokładniej po tym jej fragmencie, z którego pochodzić będą podawane w trakcie procesu samouczenia sygnały uczące. Natomiast te punkty przestrzeni wejść, które nie będą pokazywane podczas procesu uczenia, nie „przyciągną" do siebie żadnych neuronów. Dla demonstracji tego efektu przewidziałem w programie **Example 11** trzy warianty prezentacji ciągu uczącego: punkty uczące sieć mogą być pokazywane z całego widocznego obszaru wchodzącej w grę przestrzeni (taka opcja nazywa się „kwadrat"), ale mogą też być wybierane z podobszaru w kształcie krzyża albo trójkąta. Dzięki temu będziesz mógł się przekonać, że sieć naprawdę znajduje odwzorowania tylko dla tych sygnałów wejściowych, które rzeczywiście są pokazywane – w obszarach, które nie podlegają uczeniu, nie pojawi się (na ogół) ani jeden kwadracik oznaczający „warujący" w tym miejscu neuron!

Wyboru rodzaju figury, w obrębie której „rekrutowane" są punkty pokazywane sieci w każdym kolejnym etapie procesu uczenia – po ustaleniu liczby kroków, jakie program ma wykonać (parametr – **Iterations**), dokonać możesz z rozwijalnej listy **Figure** w grupie parametrów uczenia (**Teaching parame-**

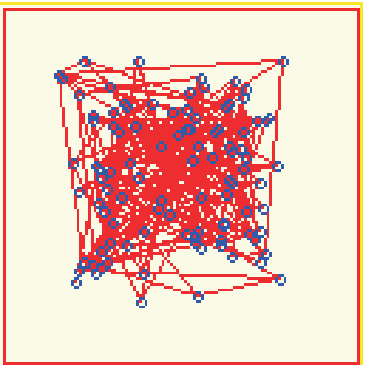
**ters**). W grupie tej znajdziesz i inne parametry procesu samouczenia, jednak na początku proponuję, byś zaakceptował ustawienia domyślne.

Teraz wystarczy już tylko przyciskać **Start,** co zapoczątkuje i będzie kontynuowało proces uczenia sieci. W okienku w lewym dolnym rogu (w obrazku obramowanym na zielono) pojawi się obraz kolejno pokazywanych sieci punktów (w liczbie, jaką określiłeś w **Iterations**). Wszystkie są w tym samym kolorze, bo oczywiście nie ma nauczyciela, który by punkty wejściowe w jakikolwiek sposób klasyfikował. Możesz natomiast zauważyć, z jakiego podobszaru przestrzeni sygnałów wejściowych pobierane były sygnały i warto porównać to z pokazanym na końcu efektem uczenia.

Po wykonaniu wskazanej przez Ciebie liczby kroków program wyświetli nową mapę, pokazującą rozkład punktów symbolizujących neurony (oraz ich sąsiedzkich powiązań, zgodnie z przyjętą topologią sieci) na tle całej przestrzeni sygnałów wejściowych. Ponieważ poprzednia mapa też jest widoczna (program pokazuje na jednym ekranie efekty kilku ostatnich etapów procesu uczenia), możesz dokładnie obejrzeć postępy, jakie sieć robi w toku procesu uczenia. Wynik ostatnio wykonanego uczenia łatwo rozpoznać po czerwonym kolorze odpowiedniej ramki; jest to potrzebne, gdy po wykonaniu wielu kroków procesu uczenia wszystkie ramki zostaną zajęte i miejsca na ekranie zaczną być wykorzystywane „rotacyjnie".

Na początku, zanim proces uczenia wprowadzi do tego jakiś porządek, neurony mają w przestrzeni wejść dość przypadkowe położenia, więc błękitne kółeczka są rozrzucone bez żadnego porządku, a linie łączące sąsiadów miotają się bez ładu i składu (rys. 10.23).

Jak stwierdziłem w poprzednim akapicie – proces losowego nadawania wartości początkowych generuje wartości z przedziału, który Ty mu podasz. Zalecałem Ci, byś do badań wykorzystywał **małe** początkowe wartości współczynników wag (na przykład proponowane przez program 0,01), więc zazwyczaj to nieuporządkowane skupisko błękitnych kwadratów symbolizują-



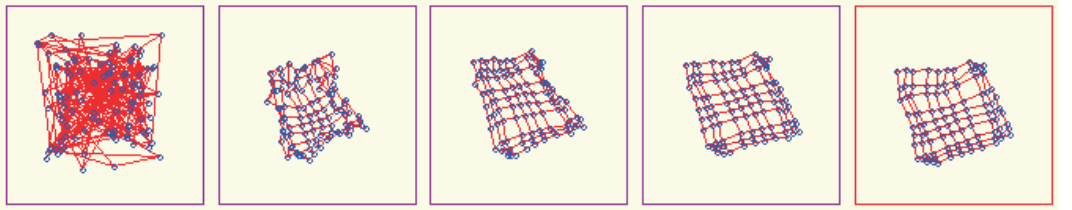
Rys. 10.23. Przykładowy obraz początkowego rozkładu wag neuronów

cych neurony zajmie gęsto wypełniony centralny obszar przestrzeni sygnałów wejściowych i będzie trudne do obejrzenia (tak jest na przykład na rysunku 10.22). Ponieważ jednak chciałem, żebyś się dokładniej przyjrzał, jak wygląda początkowy stan sieci przed rozpoczęciem procesu uczenia – przed symulacją, której początkowy etap pokazuje rysunek 10.23, podałem dużą wartość zakresu rozrzutu wstępnego losowania wag (konkretnie **7**)**.** Wiedziałem, że uczyć taką sieć będzie potem raczej trudno, ale chciałem, żebyś sam zobaczył na własne oczy, że początkowy stan sieci (a zwłaszcza początkowe położenia linii obrazujących wzajemne położenia sąsiadów) jest zupełnie chaotyczny.

Po każdorazowym pokazaniu rozkładu położeń neuronów program może zmienić ilość kroków, jakie mają być wykonane, zanim zobaczysz nową mapę rozkładu punktów. Na początku możesz skorzystać z ustawień domyślnych, a potem, gdy już dobrze „wczujesz się" w pracę programu – możesz sobie sam (według życzenia) wydłużać lub skracać odstępy pomiędzy kolejnymi prezentacjami wyników uczenia, oglądając jego przebieg w sposób bardziej efektowny (duże skoki, polegające na wykonaniu kilkudziesięciu czy kilkuset kroków uczenia i duże zmiany w obrazie działania sieci) lub dokładniejszy (małe skoki). Pod koniec procesu uczenia, gdy „postępy", jakie robi ucząca się sieć, są już dość małe, celowe będzie podawanie bardzo dużych skoków (na przykład 100 lub 300 kroków uczenia) – ale pamiętaj, że wtedy, w przypadku sieci o większym rozmiarze, będziesz musiał trochę poczekać, zanim zobaczysz wyniki!

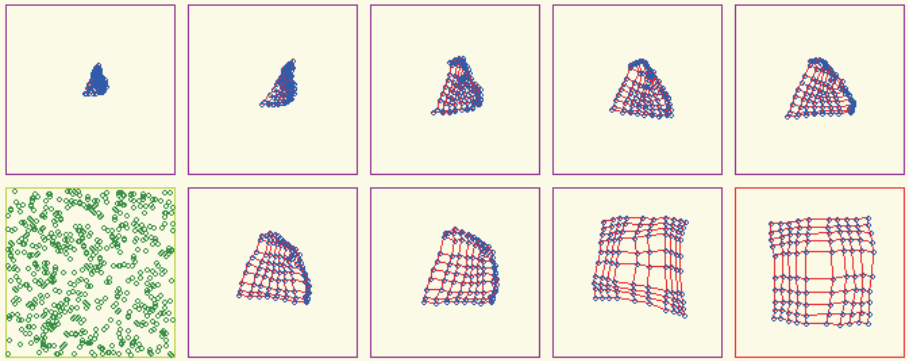
Naszym celem jest zaobserwowanie i przebadanie procesu samoorganizacji w sieci Kohonena. Prześledźmy najpierw, jak ten proces przebiega w prostych przypadkach. Jeśli uruchomisz proces uczenia i program modelujący zacznie generować i pokazywać sieci punkty pochodzące z różnych miejsc przestrzeni wejściowych sygnałów (będziesz mógł śledzić ten proces, bo w lewym dolnym rogu rysowanego przez program okna będzie widoczny obszar, w którym wyświetlane będą kolejno pokazywane sieci punkty – por. rys. 10.22), to wkrótce zaobserwujesz także przemieszczanie się niebieskich okręgów reprezentujących wagi neuronów. Okręgi te, początkowo całkowicie chaotycznie rozrzucone, stopniowo zajmować będą położenia równomiernie rozmieszczone w eksploatowanym fragmencie przestrzeni sygnałów wejściowych. Co więcej, na ich położenie będą miały wpływ relacje sąsiedztwa, co zaobserwujesz w ten sposób, że czerwone linie, sygnalizujące, który neuron jest czyim sąsiadem, stopniowo formować będą regularną siatkę. Będzie to wyglądało w taki sposób, jakby siatka, w której skład wchodzą węzły (poszczególne neurony) oraz połączenia (informujące o sąsiedztwie), podlegała rozciąganiu. W wyniku tego wektory wag neuronów warstwy topologicznej przemieszczać się będą do takich miejsc, by każdy z nich zajął miejsce, będące

centroidem (wzorcem) dla pewnego fragmentu przestrzeni sygnałów wejściowych. Dokładnie proces ten opisuje tak zwana „mozaika Voronoy'a", ale z dokładnością wymaganą w tej popularnej książce można przyjąć, że po prostu różne neurony wchodzące w skład sieci stopniowo **specjalizują się** w wykrywaniu i sygnalizowaniu różnych grup sygnałów wejściowych. W rezultacie po pewnym czasie dla każdego dostatecznie często pojawiającego się sygnału wejściowego istnieć będzie w sieci dokładnie jeden neuron wyspecjalizowany w jego wykrywaniu, sygnalizowaniu i rozpoznawaniu. W początkowym etapie (rys. 10.24) dominuje przy tym proces, w wyniku którego przypadkowy rozkład punktów i linii zastępowany jest przez wstępne uporządkowanie punktów.

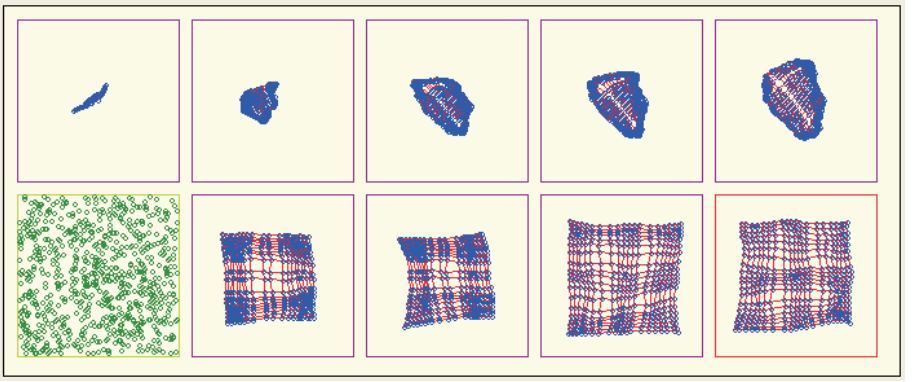


Rys. 10.24. Początkowe etapy procesu samoorganizacji, następującego w sieci Kohonena

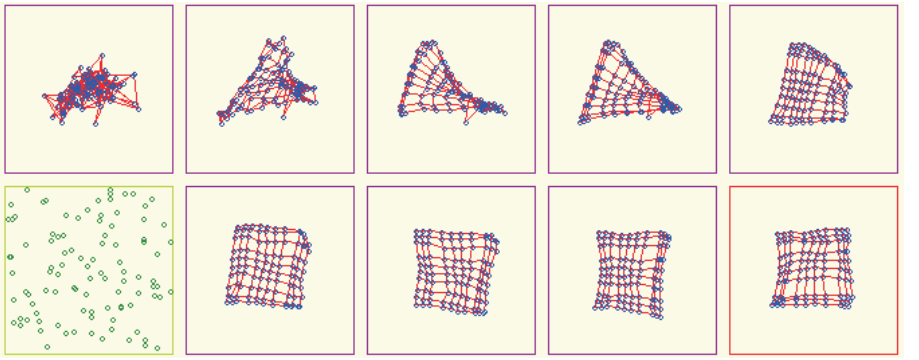
Następnie proces uczenia staje się subtelniejszy i zmierza do tego, by rozkład punktów stał się maksymalnie równomierny. Ważne jest przy tym, że sieć wytwarza wewnętrzne reprezentacje (w postaci odpowiednich rozkładów wartości współczynników wagowych) – wyłącznie dla tego podobszaru przestrzeni sygnałów wejściowych, z których pochodzą prezentowane jej



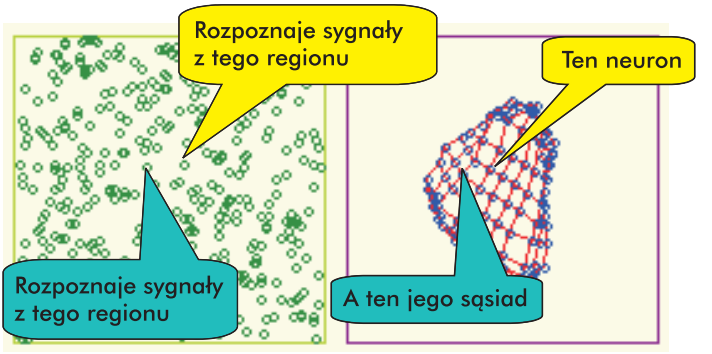
Rys. 10.25. Etapy procesu samoorganizacji, następującego w relatywnie małej sieci Kohonena, dla której dane wejściowe prezentowane są z kwadratowego podobszaru przestrzeni sygnałów wejściowych



Rys. 10.26. Etapy procesu samoorganizacji, następującego w dużej sieci Kohonena, dla której dane wejściowe prezentowane są z kwadratowego podobszaru przestrzeni sygnałów wejściowych



Rys. 10.27. Samoorganizacja sieci startująca od dużego początkowego rozrzutu wartości wag




Rys. 10.28. Efekt procesu samoorganizacji. Oglądając rysunek najpierw odczytaj komentarze wpisane na żółtych polach, a potem komentarze na polach zielonych. W obydwu przypadkach najpierw czytaj komentarze po prawej stronie, a dopiero potem te po lewej

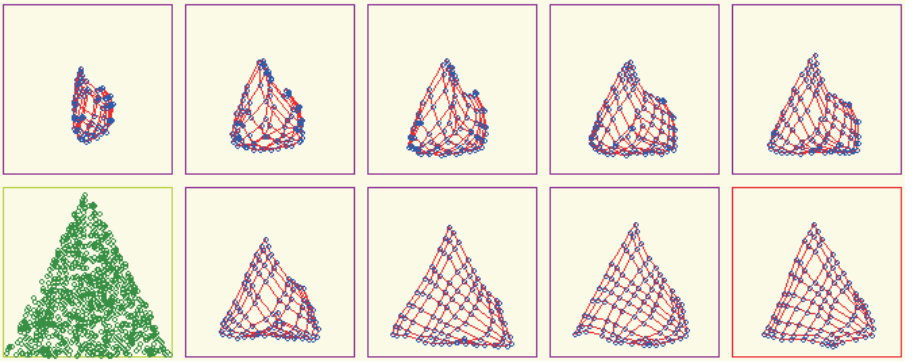
punkty. I tak możesz zaobserwować, że w przypadku gdy sygnały wejściowe pochodzić będą z ograniczonego obszaru wartości sygnałów wejściowych (na wykresie taki obszar ma formę kwadratu) – wówczas sieć Kohonena usiłuje „pokryć" neuronami możliwie dokładnie ten właśnie kwadrat. Dzieje się tak zarówno wtedy, jak masz do czynienia z siecią mającą niewiele neuronów (rys. 10.25), jak i (wolniej działającą) siecią o bardzo dużej liczbie neuronów (rys. 10.26), oraz również w przypadku, kiedy początkowy rozkład neuronów rozlokowany jest w dużym podobszarze przestrzeni wag (rys. 10.27).

Zapamiętaj, że efekt procesu samoorganizacji zawsze polega na tym, żeby dla każdego punktu przestrzeni sygnałów wejściowych znalazł się w sieci neuron, który będzie wykrywał pojawienie się tego właśnie punktu – nawet jeśli taki punkt nie był bezpośrednio pokazany w procesie uczenia (rys. 10.28).

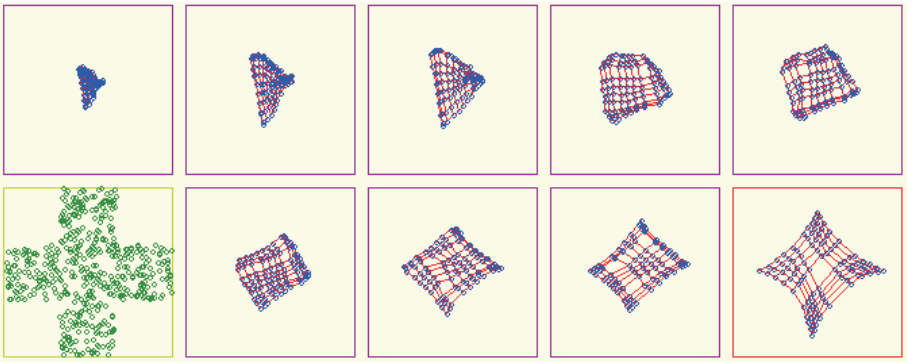
## 10.6. Co zrobią sieci Kohonena w przypadku trudniejszych danych?

Program **Example 11** ma bardzo bogate możliwości, które dadzą Ci okazję do przeprowadzenia – jeśli Ci tylko fantazji wystarczy – różnych ciekawych eksperymentów. Możesz na przykład zbadać, jak na zachowanie sieci i zachodzące w niej procesy samoorganizacji wpływać będzie sposób pokazywania danych wejściowych. Korzystając z oferowanych przez program opcji, szybko przekonasz się, że jeśli podobszar przestrzeni sygnałów wejściowych, z którego rekrutować się będą pokazywane sieci wartości tych sygnałów, zostanie jeszcze bardziej ograniczony, niż w przypadku kwadratu, to proces samoorganizacji zmierzać będzie do tego, żeby nie tworzyć nadmiarowych (zbędnych) reprezentacji wejściowych danych. Możesz to zauważyć podając sygnały wejściowe do uczącej się sieci wyłącznie z jakiegoś wybranego podobszaru przestrzeni sygnałów wejściowych, na przykład w formie trójkąta (mój program daje Ci takie możliwości). Zobaczysz wtedy, że wszystkie neurony tak się ustawią, by rozpoznawać i wykrywać wszystkie punkty wewnątrz tego trójkąta (rys. 10.29) – żaden neuron nie wyspecjalizuje się natomiast w tym, żeby rozpoznawać sygnały wejściowe z obszaru położonego poza tym trójkątem – ponieważ takie punkty w czasie uczenia nie były pokazywane, w związku z tym zapewne wcale nie występują i nie trzeba ich rozpoznawać!

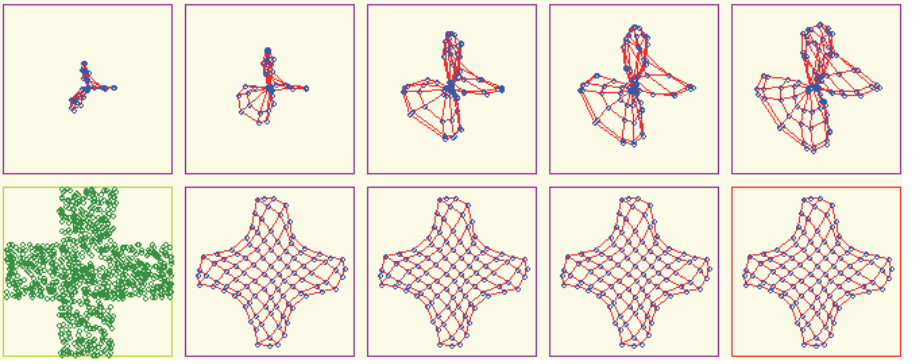
Nieco trudniej jest sieci spełnić podobne zadanie w przypadku, gdy wybrany podobszar przestrzeni sygnałów wejściowych ma bardziej złożoną formę – na przykład krzyża. W takim przypadku może się okazać, że na początku procesu uczenia sieć nie może sobie poradzić ze znalezieniem właściwego rozkładu neuronów (rys. 10.30).



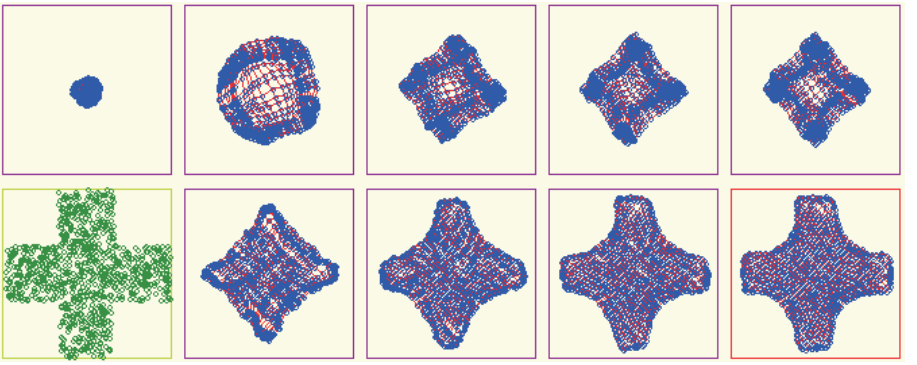
Rys. 10.29. Etapy procesu samoorganizacji, następującego w sieci Kohonena, dla której dane wejściowe prezentowane są z trójkątnego podobszaru przestrzeni sygnałów wejściowych



Rys. 10.30. Niepowodzenie procesu samoorganizacji w sytuacji, kiedy wyróżniony podobszar przestrzeni sygnałów wejściowych ma formę krzyża



Rys. 10.31. Etapy procesu skutecznej samoorganizacji, następującego w sieci Kohonena, dla której dane wejściowe prezentowane są z podobszaru przestrzeni sygnałów wejściowych w formie krzyża

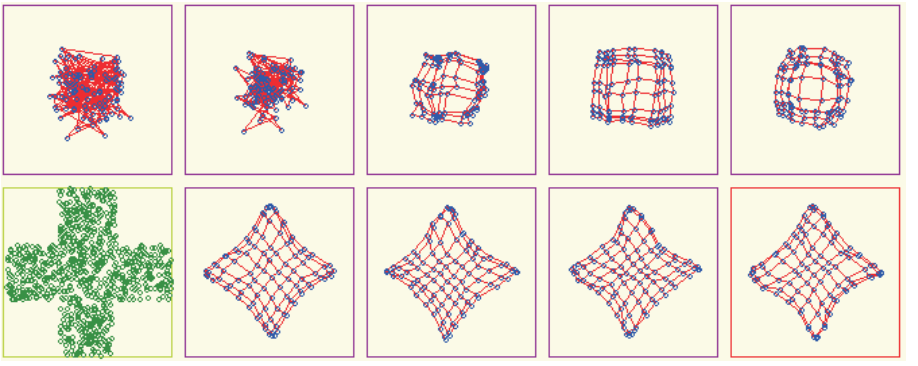


Rys. 10.32. Przykład skutecznej samoorganizacji, następującej w dużej sieci Kohonena, dla której dane wejściowe prezentowane są z podobszaru przestrzeni sygnałów wejściowych w formie krzyża

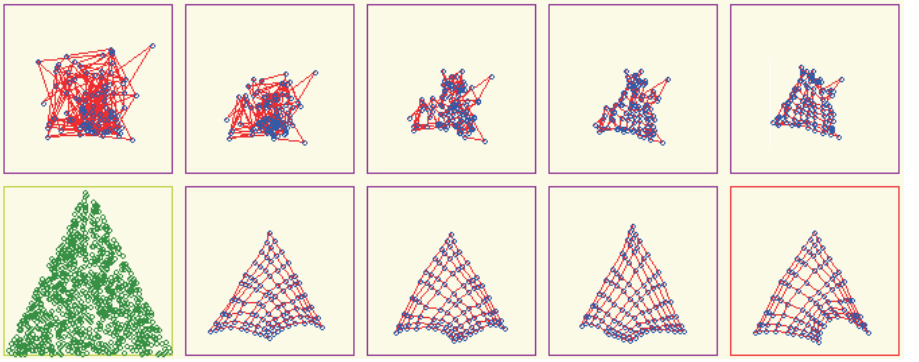
Jednak zwykle wytrwała nauka potrafi także i w tym przypadku doprowadzić do sukcesu (rys. 10.31), przy czym łatwiej jest go osiągnąć, jeśli uczenie prowadzone jest w sieci o większej liczbie neuronów (rys. 10.32).

## 10.7. Co się dzieje w sieci przy nadmiernie szerokim zakresie początkowych wag?

Duży początkowy rozrzut współczynników wagowych modelowanych neuronów jest czynnikiem zdecydowanie niesprzyjającym uzyskaniu dobrych wyników samoorganizacji (rys. 10.33). W takich przypadkach często obserwuje się także zjawisko pomijania i – mimo długiej nauki – konsekwentnego ignorowania przez samoorganizującą się sieć pewnych fragmentów aktywnych obszarów przestrzeni sygnałów wejściowych (zwróć uwagę na prawy dolny róg trójkąta na rysunku 10.34).

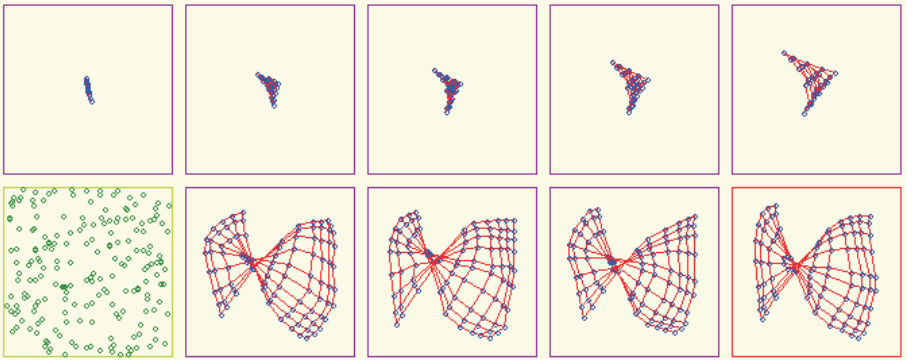


Rys. 10.33. Przykład trudności w uzyskiwaniu skutecznej samoorganizacji występujących w przypadku zbyt dużego rozrzutu początkowych wartości współczynników wag neuronów

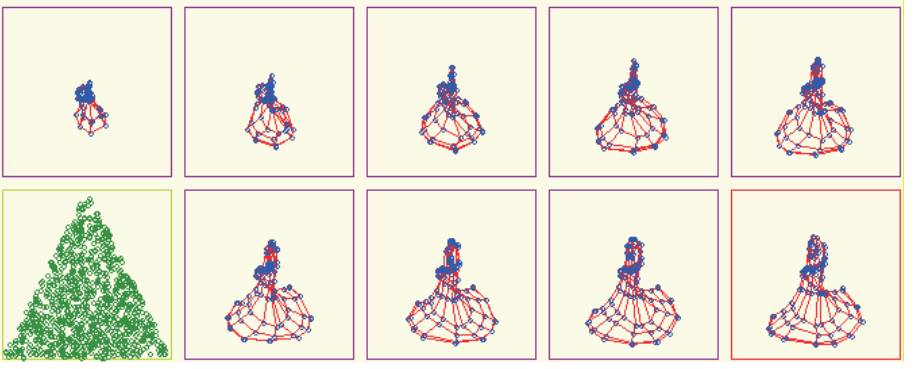


Rys. 10.34. Przykład ignorowania części danych wejściowych, występujący niekiedy w przypadku zbyt dużego rozrzutu początkowych wartości współczynników wag neuronów

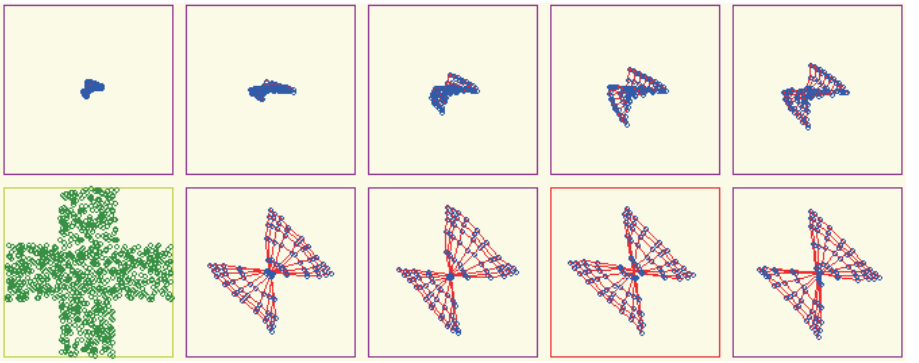
Bardzo Cię namawiam, żebyś potem, jak już obejrzysz „zdrowe" działanie sieci, „zapuścił" uczenie z bardzo dużymi wartościami początkowych losowych rozrzutów wag (na przykład wynoszącymi 5) dla jakiegoś nieskomplikowanego



Rys. 10.35. Zjawisko „skręcenia się" sieci Kohonena



Rys. 10.36. Inna forma „skręcenia się" sieci Kohonena



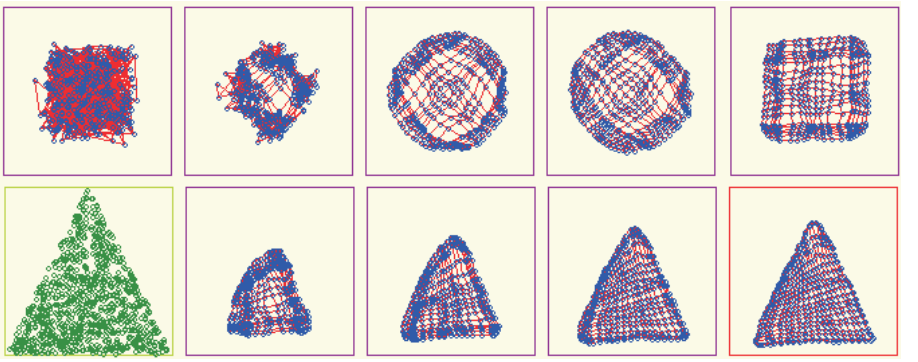
Rys. 10.37. Rzadko występujący przypadek „skręcenia się" sieci Kohonena odtwarzającej obszar w przestrzeni sygnałów wejściowych w postaci krzyża

przypadku (najlepiej dla dość małej sieci, na przykład 5 x 5, żeby szybko zobaczyć efekt). Bardzo prawdopodobne, że zobaczysz wtedy interesujące zjawisko pojawiające się niekiedy w takiej właśnie wstępnie przesterowanej sieci Kohonena, polegające na „zawinięciu" albo „skręceniu się" sieci (trudno to opisać – lepiej samemu zobaczyć na rysunkach 10.35, 10.36 i 10.37). Zastanów się w wolnej chwili, co może być powodem występowania takiego błędu i dlaczego proces samouczenia nie potrafi sieci wyprowadzić z tego „ślepego zaułka"?

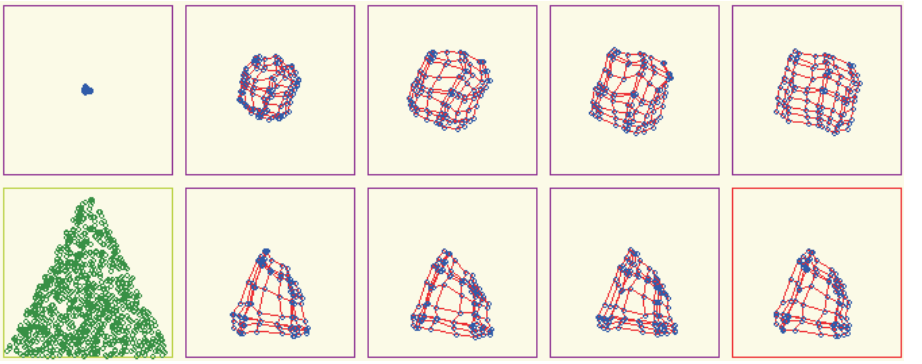
## 10.8. Czy można zmieniać formę samoorganizacji w trakcie samouczenia sieci?

Interesujące doświadczenia możesz wykonać z programem **Example 11** dzięki temu, że daje on możliwość zmiany „w biegu" formy podobszaru przestrzeni cech, z którego rekrutowane są odpowiednie dane wejściowe. Możesz na przykład rozpocząć proces uczenia z ukierunkowaniem na rekrutację danych z obszaru kwadratu, a potem, gdy już sieć prawie osiągnie stan pożądany – możesz zmienić zdanie i zażądać dostosowania się sieci do obszaru danych wejściowych w postaci trójkąta. Na rysunku 10.38 możesz obejrzeć wynik takiego eksperymentu.

Nie zawsze uda się taki zabieg przeprowadzić tak całkiem bezboleśnie – czasem w końcowym układzie sieci odzywa się ślad wcześniej wyuczonej figury. Widać to na rysunku 10.39, na którym pokazałem przebieg szczególnie perwersyjnego uczenia: najpierw sieć dopasowywała swoje działanie do obszaru w kształcie trójkąta, potem zmuszono ją do poszukiwania rozkładu o formie kwadratu, by na końcu znowu dostosować jej działanie do funkcjonowania z danymi pochodzącymi z obszaru w kształcie trójkąta.



Rys. 10.38 Zmiana celów samoorganizacji w trakcie uczenia



Rys. 10.39. Czasem zmiana celu uczenia w trakcie nauki pozostawia trwałe ślady

Jak widzisz z przytoczonych przykładów, w sieci Kohonena neurony uczą się (całkiem same, bez jakiejkolwiek interwencji nauczyciela!) odwzorowywać w swojej pamięci wewnętrznej (reprezentowanej przez współczynniki wagowe) typowo pojawiające się wzorce zewnętrznych (wejściowych) sygnałów.

## 10.9. No dobrze, tylko do czego się to wszystko może przydać?

Sądzę, że już masz pogląd, co potrafi zrobić sieć Kohonena. Powraca jednak ważne i konkretne pytanie – do czego tego można w praktyce użyć? Na początku rozdziału mówiłem Ci już o odwzorowaniach – na przykład w robotyce – które sieć Kohonena mogłaby spontanicznie tworzyć. Wiedząc teraz, jak sieć działa, możesz ten przykład przeanalizować nieco dokładniej. Wyobraź więc sobie takiego robota, który ma dwa sensory (bo sieci, które badałeś

‚miały zawsze tylko dwa sygnały wejściowe). Niech jeden z tych sensorów dostarcza informacji o wielkości oświetlenia, a drugi – o natężeniu dźwięku. W ten sposób każdy punkt przestrzeni sygnałów wejściowych odpowiadać będzie jednemu środowisku o jakimś konkretnym zestawie cech – jedne będą jasne i ciche, inne ciemne i głośne itp.

– Że co? Że coś Ci to przypomina?

– No i bardzo dobrze, właśnie ma Ci przypominać!

Otóż robot wyposażony w sieć Kohonena zaczyna swoje działanie od tego, że obserwuje otoczenie. Czasem jest w tym otoczeniu jasno, czasem ciemno, czasem głośno, czasem cicho – ale okazuje się, że jedne kombinacje sygnałów wejściowych występują w otoczeniu robota – a inne nie. Robot spokojnie klasyfikuje sobie nadchodzące dane, gromadzi o nich wiedzę, **wyspecjalizowuje** swoje neurony – aż po pewnym czasie ma już wytrenowaną swoją sieć Kohonena, w której każdej zdarzającej się w praktyce sytuacji odpowiada neuron, który ją identyfikuje i wykrywa.

Warto się przez chwilę zastanowić, co to oznacza. W istocie to, co powstaje w sieci Kohonena pełniącej funkcję „mózgu" robota – jest to *wewnętrzny model zewnętrznego świata*. **Ty także masz taki model**. Jest w nim neuron rozpoznający twarz Twojej Matki, jest neuron związany z identyfikowaniem drogi do Twojego domu, jest neuron, który rozpoznaje Twoje ulubione ciastka i taki, który sygnalizuje Ci obecność najbardziej znienawidzonego psa sąsiadów, który już dwa razy Cię ugryzł. Do każdego wrażenia zmysłowego, które rozpoznajesz, do każdej sytuacji, którą znasz – masz w mózgu model, który tę sytuację wykrywa i rozpoznaje. Wybitny polski neurofizjolog prof. Konorski związał te wewnętrzne modele określonych fragmentów zewnętrznego świata z wyodrębnionymi obszarami ludzkiego mózgu i nazwał je *jednostkami gnostycznymi.*

Praktycznie cała Twoja percepcja, cała zdolność poznawania i rozpoznawania stanu otaczającego świata oparta jest na tym, że w mózgu wytworzyłeś sobie podczas lat obcowania z różnymi sytuacjami gotowe wzorce odpowiednich wrażeń – właśnie te jednostki gnostyczne. Sygnały, jakich dostarczają Twoje oczy, uszy, nos i inne zmysły, służą w tej sytuacji zazwyczaj do tego, by uaktywnić odpowiednią jednostkę gnostyczną, czyli by z tysięcy modeli, które masz przygotowane w mózgu – wybrać i uruchomić ten jeden, odpowiadający aktualnie napotkanemu wrażeniu albo sytuacji, w jakiej się aktualnie znalazłeś. Dzięki temu i – jak się okazuje – wyłącznie dzięki temu, Twoje rozpoznawanie wrażeń zmysłowych jest tak szybkie, sprawne i niezawodne. Jeśli jednak w mózgu brakuje takiego gotowego, wcześniej ukształtowanego modelu, wówczas sytuacja percepcyjna staje się trudna, a orientacja w nowej sytuacji bywa bardzo opóźniona i często zawodna.

Istnieją setki powodów, żeby nie rozwlekać tego wątku i pozwolić Ci jak najszybciej znowu pobawić się siecią neuronową – wspomnę tylko o trzech. Najbardziej ważkie dowody, że opisane wyżej wewnętrzne modele zewnętrznego świata istnieją i są nieodzowne, uzyskano wykonując doświadczenia na zwierzętach. Prowadzono na przykład deprywację młodych kotów[1]. Polega ona na tym, że gdy młody kot zaczyna widzieć na oczy (jak wiadomo kocięta rodzą się ślepe i zdobywają zdolność widzenia dopiero po kilku dniach) – pozwala mu się oglądać wyłącznie jakiś jeden ustalony wzorzec geometryczny – na przykład tylko pionowe pasy. We wszystkich okolicznościach, kiedy kot mógłby zobaczyć coś innego (na przykład podczas karmienia) – wyłącza się światło. Po miesiącu takiego treningu wypuszcza się kota, pozwalając mu żyć w normalnym świecie. Okazuje się, że kot, mając zupełnie zdrowe i w pełni sprawne oczy – zachowuje się jak niewidomy. Nie jest w stanie wzrokiem zauważyć przeszkody, miski z mlekiem czy nawet człowieka. Jego mózg dysponuje wyłącznie modelami prostych figur geometrycznych, nie ma natomiast w nim miejsca na wzorzec krzesła, garnka czy innego kota. Jego percepcja jest całkowicie zakłócona i potrzeba długiej nauki, żeby kot odzyskał zdolność normalnego widzenia.

O tym, że podobne zjawiska zachodzą także u ludzi, świadczą informacje antropologów, którzy opisywali przygody Pigmejów. Jest to plemię afrykańskich karłów, których naturalnym środowiskiem jest wnętrze gęstej dżungli, w którym nie ma możliwości patrzenia na odległość. Otóż ludzie ci wyprowadzeni na otwartą przestrzeń całkowicie tracili zdolność orientacji. Najprostsze zjawiska związane z patrzeniem w dal, banalne dla każdego z nas, takie jak zmiany perspektywy związane na przykład ze zbliżaniem się jakiegoś zwierzęcia – traktowali jako wynik niezrozumiałych i potężnych czarów (*przecież jeszcze przed chwilą ta zebra była mniejsza od psa, a teraz urosła do rozmiarów konia!*). Brakowało w ich mózgach *modeli* związanych z percepcją wrażeń wzrokowych na otwartej przestrzeni.

Czasem o znaczeniu wewnętrznych modeli obiektów rozpoznawanego przez Ciebie świata możesz się przekonać osobiście – jeśli tylko zwrócisz uwagę na proste i na pozór oczywiste zjawiska. Na przykład, bez trudu czytasz wiadomości na dowolny temat w języku polskim, a skoro jesteś informatykiem, to zapewne także w angielskim, być może niemieckim, francuskim, hiszpańskim itd. W każdym z tych języków, jeśli tylko znasz go dostatecznie dobrze, wystarczy rzut oka – i już w mózgu litery łączą się w słowa, ze słów formują się pojęcia, z pojęć wiedza... Dzieje się tak dlatego, że w Twoim mózgu przez wiele lat nauki uformowały się modele liter, słów, zdań i całych

[1] Nie **deprawację** tylko **deprywację,** ty zbereźniku!

wiadomości, w związku z czym czytając „wywołujesz" je tylko, a one natychmiast pojawiają się – gotowe do użytku.

Odmienna sytuacja pojawia się, gdy napotykasz w tekście nieznane słowo, albo wręcz próbujesz przeczytać wiadomość w nieznanym języku. Wtedy z wysiłkiem sylabizujesz tekst, bardzo dużo czasu zajmuje Ci jego przeliterowanie, nie mówiąc już o całkowitej niemożności zrozumienia sensu zawartego w podanych wypowiedziach. Łatwość i szybkość czytania znikają bezpowrotnie, bo w Twoim mózgu nie ma gotowych wzorców, nie ma *modeli* dla tych słów, zdań i wiadomości.

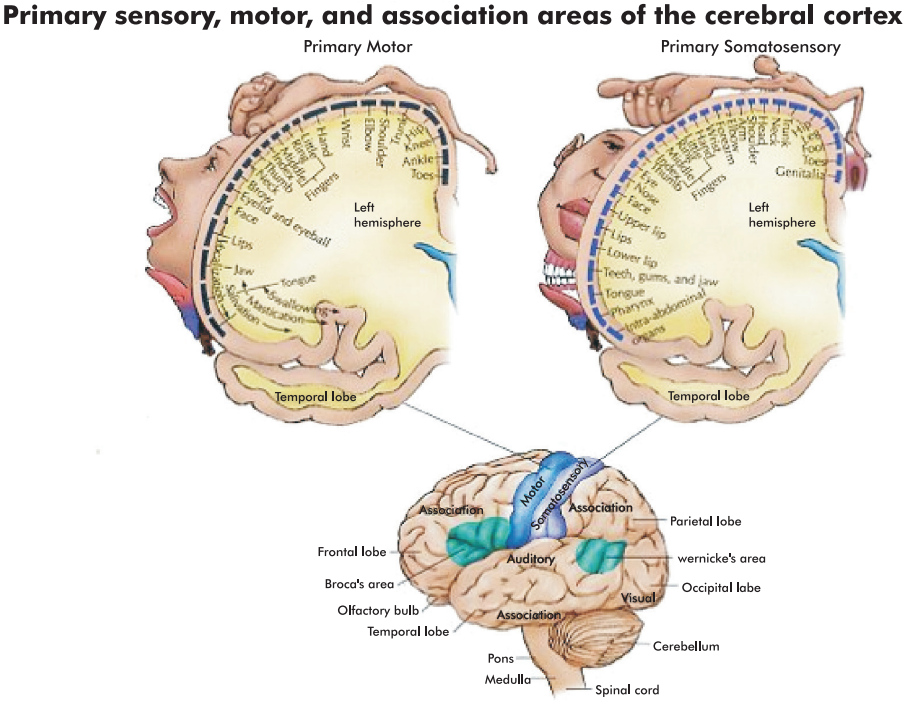
Mógłbym rozszerzyć ten wątek, pokazując, że w przypadku napisów formułowanych w nieznanym języku (na przykład po węgiersku), ale wykorzystujących znajomy alfabet – możliwe jest skorzystanie ze znajdujących się w mózgu wzorców liter, dzięki którym możesz nieznany napis zapamiętać i spotkawszy tłumacza odtworzyć go prosząc, by Ci go przetłumaczył. Jeśli jednak nawet wzorców potrzebnych liter nie masz w swoim wewnętrznym modelu zewnętrznego świata (na przykład, gdy spotykasz się z nieznanym napisem w Japonii albo w Iranie) – wówczas nie tylko nie zdołasz zrozumieć treści napisu, ale także niesłychanie trudno będzie Ci go zapamiętać i odtworzyć po spotkaniu z tłumaczem w celu uzyskania pomocy. Po prostu, Twój mózg nie będzie miał w zapasie żadnych użytecznych w tym przypadku jednostek gnostycznych.

W Twoim mózgu znajduje się także inne odwzorowanie, które tworzy „mapę" całego Twojego ciała na powierzchni kory mózgowej w obszarze zwojów nazywanych *gyrus postcentralis* (rys. 10.40). Co ciekawe: ta mapa wcale nie przypomina dokładnie kształtów i proporcji Twojego ciała! Na powierzchni mózgu na przykład dłonie i twarz zajmują znacznie więcej miejsca niż cały korpus wraz z kończynami (rys. 10.41)!

Wynika to właśnie z tego faktu, o którym wspomniałem wyżej w kontekście sztucznych samoorganizujących się sieci neuronowych: sterowanie ruchami dłoni i twarzy (mimika, mowa), a także odbiór bodźców sensorycznych z tych właśnie obszarów angażuje znacznie więcej komórek mózgu, bo czynności te wykonywane są **często.**

Przepraszam, że tak się rozpisałem na temat tych przykładów, zależało mi jednak na tym, żebyś zrozumiał i zapamiętał, jak wielkie i doniosłe znaczenie mają w procesie widzenia (czy dowolnej innej percepcji: słyszenia, smaku czy nawet dotyku) wcześniej przygotowane w mózgu wzorce przyswajanych wrażeń. I właśnie takie wzorce tworzą się całkowicie spontanicznie w sieciach neuronowych w trakcie procesu samoorganizacji.

Wróćmy teraz do rozpoczętego kilka zdań wcześniej przykładu z robotem. Każdy neuron sieci Kohonena tego robota, rozpoznający jeden ze **zdarzają-**




Rys. 10.40. Reprezentacja w korze mózgowej człowieka poszczególnych części ciała w kontekście funkcji sterowania ruchem (po lewej) oraz odbioru i analizy bodźców czuciowych (po prawej)

Rys. 10.41. Obraz człowieka, na którym poszczególne części ciała mają rozmiary proporcjonalne do wielkości ich reprezentacji w mózgu

**cych się** (tylko takich!) stanów zewnętrznego świata, pełni w systemie sterowania robota dokładnie tę samą funkcję, jak opisana wyżej jednostka gnostyczna w Twoim mózgu. Zatem proces samouczenia sieci Kohonena tworzy w sterowniku robota także swoisty zespół jednostek gnostycznych, będących zbiorem modeli dla każdej sytuacji, w jakiej może się znaleźć wykorzystujący sieć robot. Taki model jest bardzo ważny, ponieważ mając model można z jego pomocą sklasyfikować każdą sytuację wejściową (wynikającą z sygnałów dostarczanych przez sensory). Z kolei zaklasyfikowawszy aktualną sytuację do jednej z wcześniej ustalonych klas, robot może odpowiednio dostosować do tej sytuacji swoje zachowanie. W najprostszym przypadku Ty sam możesz ustalić, co robot ma robić w pewnych konkretnych sytuacjach – na przykład możesz ustalić, że jeśli sytuacja jest normalna (cicho i niezbyt jasno), to robot powinien się posuwać do przodu, jeśli światło zgaśnie, powinien się zatrzymać, jeśli usłyszy hałas, to powinien się cofnąć itp. Ważne jest, że nie musisz wcale podawać szczegółowych wytycznych, co robot ma robić w **każdej** możliwej sytuacji. Na podstawie **sąsiedztwa** neuronów wykrywających **podobne** sytuacje możesz poprzestać na podaniu tylko kilku wskazówek dla kilku wybranych sytuacji, a wytrenowana sieć Kohonena w „mózgu" robota sama ustali, co należy zrobić. Po prostu stwierdziwszy, że w stosunku do aktualnej sytuacji rejestrowanej przez jego sensory nie ma gotowej recepty postępowania, robot sam rozpozna (na podstawie struktury sąsiedztwa w sieci neuronowej), że spośród stanów sieci, dla których określono najwłaściwsze działanie, **najbliższa** jest pewna inna konkretna sytuacja. Jeśli dla tej najbliższej sytuacji zapisano w pamięci, że należy zrobić coś konkretnego – to zapewne w aktualnie rozpoznanej podobnej sytuacji celowe jest zrobienie tego samego albo czegoś podobnego. Na przykład jeśli sytuacja, stanowiąca wykorzystywaną analogię, związana była z poleceniem szybkiej jazdy do przodu – we wszystkich nie zdefiniowanych przez użytkownika sytuacjach „sąsiednich" można też zastosować jazdę do przodu, ale na przykład ze zredukowaną prędkością.

W ten sposób jeśli dla kilku tylko modelowych sytuacji określisz, co i kiedy należy robić – robot będzie umiał (lepiej albo gorzej) zachować się w **dowolnej** możliwej sytuacji. Warto podkreślić, że dotyczy to nie tylko ściśle i dokładnie tych obiektów, które pojawiły się w trakcie procesu samouczenia sieci. Podczas samouczenia sieci Kohonena masz do czynienia – jak to zwykle w sieciach neuronowych bywa – z procesem **uśredniania** i **generalizacji**.

W związku z procesem uśredniania warto odnotować jeszcze jeden fakt. Otóż podczas uczenia mogą być pokazywane obiekty (środowiska) nieco różniące się od siebie, czyli charakteryzujące się pewnym **rozrzutem** – a jednak zapamiętany dla nich zostanie (w postaci zestawu wartości współczynników wag odpowiednich neuronów) pewien wypadkowy, uśredniony wzorzec „ty-

powego wejściowego sygnału", który sieć wyznaczy sobie (całkiem sama!) w toku procesu nauki. Tych typowych wzorcowych sygnałów (i związanych z nimi modelowych środowisk) będzie oczywiście dużo – dokładnie tyle, ile neuronów liczy używana sieć. Jednak i tak będzie ich znacznie mniej niż możliwych środowisk, ponieważ przy dowolnie zmieniających się parametrach charakteryzujących sytuację, w jakiej znalazł się robot – możliwych środowisk jest nieskończenie dużo!

Z kolei dzięki generalizacji w trakcie „egzaminu" (czyli podczas normalnej eksploatacji robota) sieć może znaleźć się w środowisku o takich parametrach, jakie nigdy nie były pokazywane w czasie uczenia. Jednak każdy neuron sieci, nawet stykając się po raz pierwszy z pewnymi sygnałami, usiłuje je zakwalifikować do tej grupy, której wypracowany podczas uczenia wzorzec jest najbardziej podobny do aktualnie rozważanego sygnału. Powoduje to, że wiedza zdobyta przez sieć podczas uczenia jest w trakcie eksploatacji sieci automatycznie uogólniana – co często daje bardzo dobre wyniki.

Opisany wyżej przykład z robotem, który dzięki sieci Kohonena dostosowuje swoje zachowanie do środowiska przemawiał (mam nadzieję!) do Twojej wyobraźni, ale na co dzień ludzie nie potrzebują zbyt wielu inteligentnych mobilnych robotów, więc gdybyś miał się utrzymywać z budowy sieci Kohonena dla takich robotów – to zapewne nie byłoby Cię stać na willę z basenem i wytworny samochód – na co z pewnością zasługujesz. Tak się bowiem składa, że nawet najgenialniejsze systemy techniczne nie przynoszą ich twórcom pieniędzy – jeśli nie ma chętnych do tego, by je zakupić i zastosować. Być może wytworzyłeś już sobie model tej sytuacji w swoim mózgu? Jeśli tak, to pomyślmy – kto i do czego **naprawdę** może zastosować sieci Kohonena?

Wszystkich pomysłów Ci tu nie streszczę, bo jest ich zbyt wiele, ale kilka ciekawszych – żebyś miał od czego zacząć własne przemyślenia – to proszę bardzo. W poprzednim akapicie była mowa o pieniądzach. Pieniądze, jak wiadomo są w bankach, a banki bywają okradane...

Nie, nie namawiam Cię, żebyś zbudował mobilnego robota sterowanego siecią Kohonena, który znalazłszy się przed okienkiem kasowym będzie metalicznym głosem mówił: *Ręce do góry! To jest napad! Dawać forsę!* Namawiam Cię jednak, żebyś pomyślał o możliwościach, jakie tkwią w sieciach Kohonena w związku z **zabezpieczeniem** banku przed kradzieżą. I to nie taką kradzieżą w stylu filmowym – z zamaskowanymi bandytami, strzelaniną i ucieczką szybkim samochodem po krętych ulicach. Takie kradzieże zdarzają się (na szczęście) rzadko, a Policja ma już swoje sposoby na to, żeby im przeciwdziałać. Natomiast prawdziwym problemem banków są dzisiaj kradzieże wykonywane w białych rękawiczkach – na przykład poprzez wyłudzanie kredytów i niespłacanie ich.

Bank z tego żyje, że pożycza pieniądze, więc nie może wyrzucać za próg każdego, kto się zgłosi z wnioskiem o kredyt – chociaż to by gwarantowało 100% bezpieczeństwa. Jednak część pożyczkobiorców nie spłaca potem pożyczonych pieniędzy – i to powoduje straty, zwykle o wiele większe niż osławione napady z bronią w ręku.

Jak poznać, komu można bezpiecznie pożyczyć pieniądze, a komu nie?

To proste – trzeba mieć **model** uczciwego przedsiębiorcy, który za zaciągnięty kredyt zbuduje dobrze prosperujący zakład i spłaci bankowi należność z procentem, oraz trzeba mieć **model** naciągacza, który dąży tylko do zagarnięcia pieniędzy i zniknięcia wraz z nimi w tak zwanej sinej dali. Niestety, ludzie są bardzo różni i różne są sytuacje życiowe i ekonomiczne. Dlatego modeli uczciwych przedsiębiorców muszą być tysiące i przynajmniej tyle samo (a może więcej?) potrzeba modeli naciągaczy i malwersantów. Żaden człowiek nie podoła takiemu zadaniu.

Od czego jednak są możliwości sieci neuronowej? Wystarczy tylko dobrze przemyśleć, jakie dane do niej wprowadzić, jak przeprowadzić samouczenie, jak interpretować potem wyniki – i po sprzedaniu kilku bankom tego genialnego systemu już można się wybierać na zasłużone wakacje na słoneczne plaże Karaibów.

– Że co, za mało danych? Że powinienem dokładniej opisać, jak to zrobić?

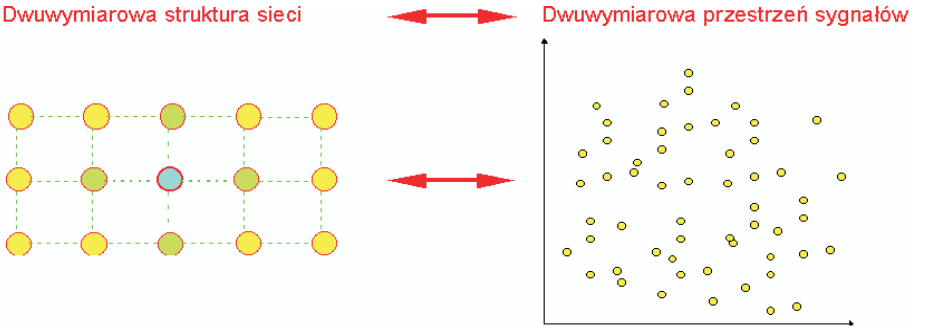
– A czy wiesz, jaki tłok by się zrobił na koralowych plażach Puerto Rico, gdybym podał tu wszystkie szczegóły i każdy, kto przeczyta tę książkę, zostałby milionerem? Musisz trochę sam ruszyć głową!

## 10.10. W jaki sposób sieć może służyć jako narzędzie do transformacji wymiaru przestrzeni sygnałów wejściowych?

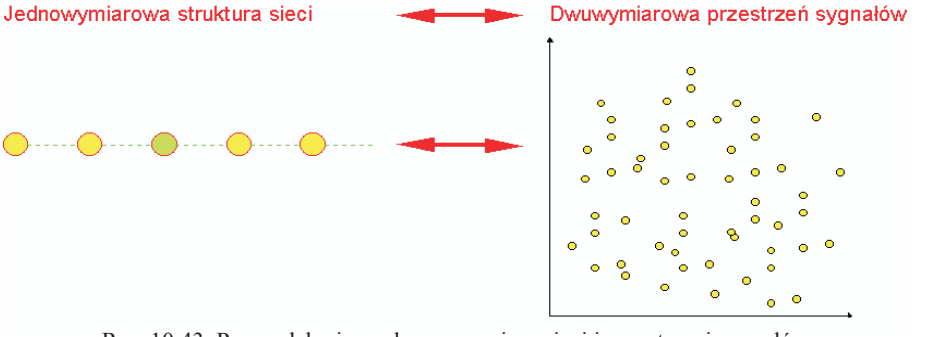
Po tej krótkiej wycieczce w kierunku błękitnego morza i szumiących palm wróćmy jednak teraz do dalszych zagadnień naukowych. Unikatową cechą sieci Kohonena jest występowanie w niej **topologicznej reprezentacji przestrzeni sygnałów wejściowych** w odpowiedziach sieci. Chodzi oczywiście o sąsiedztwo i jego konsekwencje. Na rysunkach, które oglądałeś wyżej i na tych, które zobaczysz podczas samodzielnej eksploatacji opisanego programu, poszczególne błękitne punkty reprezentujące neurony łączone są ze sobą czerwonymi liniami. Linie te, jak wiesz, łączą ze sobą neurony uważane za sąsiednie. Na początku linie te – podobnie jak same punkty – ułożone są czysto przypadkowo. Podczas uczenia zauważasz jednak, że sieć ucząc się zacznie wytwarzać **regularną siatkę** złożoną z czerwonych linii. Czym w istocie jest ta siatka? Otóż jest ona wyrazem prawidłowości polegającej na tym, że **są-**

**siednie neurony** będą dążyły do tego, żeby sygnalizować i wykrywać **sąsiadujące ze sobą punkty** w przestrzeni sygnałów wejściowych! W ten sposób porządek wynikający z tego, że niektóre punkty odpowiadające sygnałom wejściowym są bliskie sobie, bo ich współrzędne różnią się w niewielkim stopniu – zostanie przeniesiony do sieci, w której bliskie punkty będą sygnalizowane wyłącznie przez sąsiadujące ze sobą neurony.

Na wszystkich dotychczas oglądanych obrazkach dwa pojęcia: bliskości (podobieństwa) sygnałów wejściowych i bliskości (sąsiedztwa) neuronów w sieci mogły być ze sobą w dosyć naturalny sposób wiązane i kojarzone, ponieważ przestrzeń sygnałów wejściowych (i oczywiście także przestrzeń wag) była dwuwymiarowa (jak na rysunku 10.13 albo 10.16) i równocześnie także topologia sieci była dwuwymiarowa (jak na rysunku 10.9 albo 10.11). Istniała więc naturalna odpowiedniość pomiędzy takimi pojęciami, jak *„sygnał wejściowy leżący wyżej od poprzedniego”* (w tym sensie, że miał większą wartość drugiej składowej) i *„neuron wejściowy leżący wyżej od poprzedniego”* (w tym sensie, że leży on – według umownie przyjętej numeracji – w poprzednim wierszu – rys. 10.42).



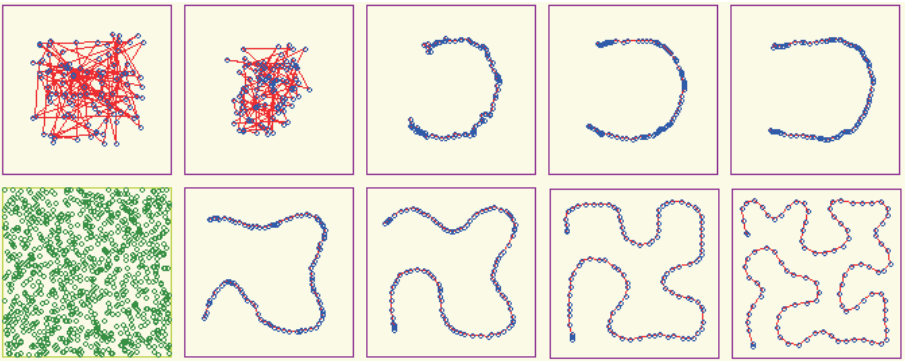

Rys. 10.42. Przypadek zgodnego wymiaru sieci i przestrzeni sygnałów



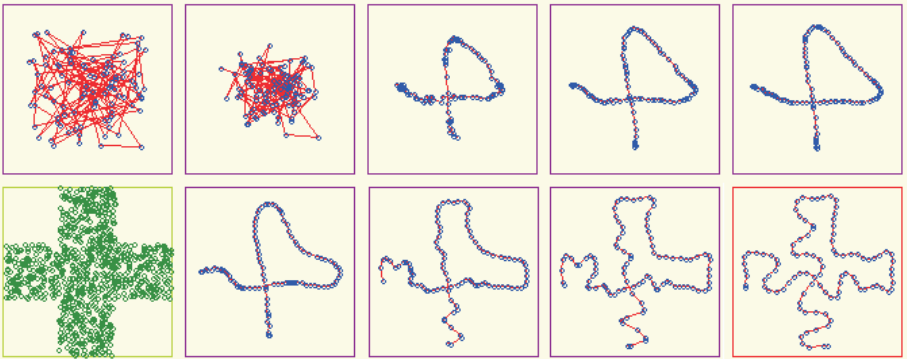

Rys. 10.43. Przypadek niezgodnego wymiaru sieci i przestrzeni sygnałów

Taka sytuacja nie jest jedyną możliwą. Można sobie równie dobrze wyobrazić sieć jednowymiarową, która będzie uczyła się rozpoznawać sygnały dwuwymiarowe (rys. 10.43).

Możesz przebadać, jak zachowuje się sieć dokonująca takiej swoistej konwersji dwuwymiarowej przestrzeni sygnałów wejściowych do jednowymiarowej struktury sieci, ponieważ mój program pozwala Ci tworzyć **sieci jednowymiarowe** (łańcuchy neuronów).



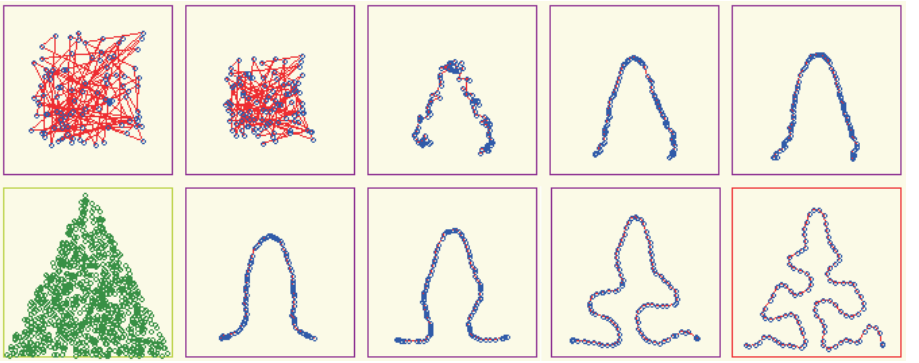
Rys. 10.44. Odwzorowanie dwuwymiarowej przestrzeni wejść w jednowymiarowej sieci



Rys. 10.45. Bardziej złożony przykład odwzorowania dwuwymiarowej przestrzeni wejść w jednowymiarowej sieci

Taką nietypową strukturę sieci możesz uzyskać podając jeden z wymiarów sieci (najlepiej pierwszy) jako równy 1. Drugi wymiar warto wtedy dać duży, na przykład 100, bo wtedy sieć ciekawiej się zachowuje, a uczy się toto dość szybko, więc nie trzeba się (w tym przypadku) nadmiernie ograniczać.

Spójrz na rysunki 10.44, 10.45 i 10.46. Widać na nich, że jednowymiarowa sieć dość sensownie **wpisuje się** w wyróżniony obszar przestrzeni sygnałów wejściowych – gwarantując dwie dość ważne zalety.

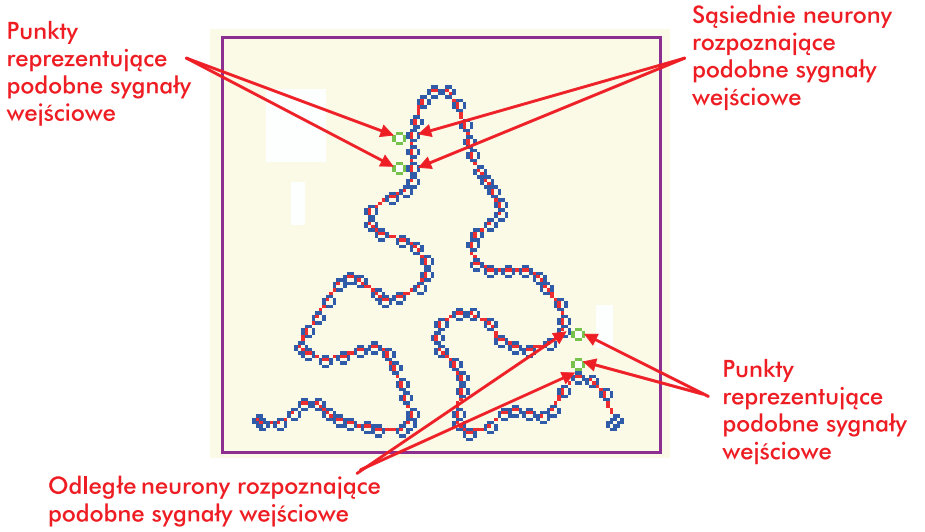


Rys. 10.46. Jeszcze jeden przykład odwzorowania dwuwymiarowej przestrzeni wejść w jednowymiarowej sieci

- Po pierwsze, łańcuch neuronów, będący jednowymiarową siecią, wypełnia (w lepszy lub gorszy sposób) cały wybrany obszar przestrzeni sygnałów wejściowych. Oznacza to, że dla każdego punktu **dwuwymiarowej** przestrzeni sygnałów wejściowych jest w **jednowymiarowej** sieci neuronowej jego reprezentant, który będzie sygnalizował jego pojawienie się. Nie ma w wielowymiarowej przestrzeni wejściowej punktów czy obszarów „bezpańskich".
- Po drugie, obiektom, które w przestrzeni sygnałów wejściowych są położone blisko siebie (a więc zapewne są w jakimś sensie podobne do siebie nawzajem) – odpowiadają w jednowymiarowym łańcuchu neuronów sąsiadujące ze sobą neurony. Niestety, tak jest **na ogół, ale nie zawsze**, trzeba się więc liczyć z możliwością pomyłek (rys. 10.47), jednak w znaczącym odsetku punktów przestrzeni sygnałów wejściowych fakt, że jakieś dwa stany wejść są sygnalizowane przez sąsiednie neurony, upoważnia do twierdzenia, że są to stany podobne.

Zastanówmy się wspólnie, jakie to może mieć znaczenie. Otóż w ogromnej liczbie zadań związanych z informatyką spotykamy się z podobnym i bardzo trudnym problemem. Dla uzyskania orientacji w pewnych zjawiskach i w pewnych procesach gromadzone są ogromne ilości danych. Na przykład, żeby zorientować się w aktualnym stanie reaktora jądrowego w dużej elektrowni, trzeba mierzyć i oceniać setki różnych parametrów. To samo dotyczy wielkiego pieca w hucie, wielosilnikowego samolotu podczas startu, a także dużego przedsiębiorstwa, którym chcemy sprawnie zarządzać. Aktualny stan każdego z tych wielkich systemów opisywany jest więc przez setki i tysiące parametrów liczbowych, możemy zatem wyobrażać go sobie jako punkt w przestrzeni o wielkiej liczbie wymiarów. Ta wielowymiarowa przestrzeń jest potrzebna, bo wynik każdego pomiaru, efekt każdej obserwacji, stan każdego sygnalizatora – powinien być odkładany na osobnej osi. W miarę rozwoju pewnych

procesów wartości wszystkich parametrów ulegają zmianie i punkt odpowiadający stanowi rozważanego systemu zmienia swoje położenie. W związku z tym dla pełnej oceny bezpieczeństwa samolotu, stabilności reaktora, sprawności pieca czy rentowności przedsiębiorstwa – trzeba stale oceniać położenie punktu w wielowymiarowej przestrzeni. Konkretnie – chodzi o położenie takiego punktu reprezentującego aktualny stan w stosunku do obszarów, którym potrafimy przypisać określone znaczenie praktyczne: stabilna praca reaktora – albo symptomy jego przegrzania, dobra jakość wytwarzanej surówki – albo jej wadliwy skład, korzystny rozwój przedsiębiorstwa – albo widmo bankructwa.




Rys. 10.47. Przykłady poprawnego oraz niepoprawnego odwzorowania relacji zachodzących pomiędzy obiektami w wielowymiarowej przestrzeni, reprezentowanymi przez neurony w sieci o niskiej wymiarowości

Chcąc kontrolować i nadzorować rozważany proces, musimy dysponować aktualną i pewną informacją o tym, jaki jest jego stan, w jakim kierunku się zmienia, ku czemu zmierza. Próbując rozstrzygać tego typu problemy we współczesnej technice, najczęściej stosujemy rozwiązania polegające na pieczołowitym zbieraniu i udostępnianiu wszystkich oryginalnych danych osobie, która ma podejmować decyzje. Efektem są znane Ci pewnie z filmów olbrzymie sterownie elektrowni jądrowych, o ścianach upstrzonych dosłownie różnymi lampkami i wskaźnikami, kabiny pilotów naddźwiękowych odrzutowców, w których każdy centymetr wolnej przestrzeni wypełniają jakieś przyrządy, zegary i sygnalizatory, a także kilometrowe wydruki komputerowe

zawierające tabele liczb i wykresy wskaźników ekonomicznych, nad którymi biedzą się i głowią biznesmeni.

Takie rozwiązania są jednak w praktyce fatalnie nieefektywne, co wynika głównie z faktu, że żaden człowiek nie jest w stanie skutecznie nadzorować, kontrolować i analizować tysięcy danych. W dodatku tak naprawdę operatorowi reaktora jądrowego, pilotowi samolotu czy dyrektorowi firmy wcale nie zależy na szczegółowych i dokładnych danych. Potrzebna mu jest syntetyczna globalna informacja: wszystko idzie dobrze, albo – dzieje się coś złego. Taką właśnie informację potrafi wypracować sieć Kohonena.

Wyobraź sobie, że zbudowałeś sieć, w której do każdego neuronu dociera kilkaset albo nawet kilka tysięcy sygnałów, odpowiadających wszystkim zbieranym danym pomiarowym. Taka sieć wcale nie jest trudniejsza do zaprogramowania od sieci o dwóch wejściach, tylko wymagać będzie więcej miejsca w pamięci komputera i więcej czasu podczas symulacji jej działania. Wyobraź sobie także, że sieć ta przewiduje dwuwymiarowe sąsiedztwo neuronów, co wykorzystasz w ten sposób, że sygnał wyjściowy każdego neuronu wyświetlać będziesz w pewnym (z góry ustalonym) punkcie na ekranie, zaś sygnały sąsiadów wyświetlisz odpowiednio w sąsiednich wierszach lub kolumnach, żeby uwidocznić ich związek. Po nauczeniu tej sieci metodą Kohonena uzyskujesz narzędzie dokonujące swoistego „rzutowania" wielowymiarowych, trudnych do oceny danych, na płaszczyznę jednego ekranu, który łatwo objąć wzrokiem, łatwo ocenić i łatwo zinterpretować. Obraz na ekranie dostarczać będzie danych, które można interpretować w następujący sposób:

- Każdej zdarzającej się w praktyce kombinacji sygnałów wejściowych odpowiadał będzie dokładnie jeden neuron („zwycięzca"), który będzie wykrywał i sygnalizował pojawienie się tej właśnie sytuacji. Jeśli będziesz wyświetlać na ekranie sygnał wyjściowy tylko tego właśnie neuronu, to uzyskasz obraz z postaci przemieszczającego się na ekranie świetlnego punktu. Jeśli na podstawie wcześniejszych badań zapamiętasz, które obszary ekranu odpowiadają prawidłowym stanom nadzorowanego procesu, a które odpowiadają stanom zagrożenia – to obserwując, gdzie aktualnie znajduje się świetlny punkt i w jakim kierunku zmierza – możesz uzyskać syntetyczną ocenę stanu i perspektyw nadzorowanego systemu.
- Na ekranie możesz też wyświetlać sygnały wyjściowe neuronów bez dokonywania wśród nich dyskryminacji wynikającej ze stosowania zasady konkurencji. Możesz umówić się, że różnym wartościom sygnałów wyjściowych przypiszesz różne kolory i wtedy zmianom w monitorowanym procesie odpowiadać będzie wyświetlana na ekranie zmienna barwna mozaika. Niesie ona znacznie więcej informacji o stanie nadzorowanego procesu i po zdobyciu pewnej praktyki pozwala na dość wnikliwe i dokładne wnioskowanie

o dostrzeganych trendach, ale na pierwszy rzut oka może być niestety dość mało czytelna.

Sposobów zobrazowania wyników pracy sieci Kohonena może być więcej. Zwróć jednak uwagę na jedną ich wspólną cechę: są to zawsze obrazy dwuwymiarowe, wygodne do ogarnięcia jednym spojrzeniem i względnie łatwe do interpretacji. Obrazy te nie przenoszą informacji o szczegółowych wartościach konkretnych danych (do nich zawsze w razie potrzeby można sięgnąć), ale pokazują syntetyczny, całościowy obraz, który tak bardzo jest potrzebny osobie chcącej podejmować decyzje i ogólnie oceniać ich skutki. Obrazy te zmniejszają stopień szczegółowości analizowanych danych, ale dzięki zjawiskom uśredniania i generalizacji, dyskutowanym wyżej, można w ten sposób uzyskiwać naprawdę bardzo dobre wyniki.

## 10.11. Pytania kontrolne i zadania do samodzielnego wykonania

1. Jaka jest różnica pomiędzy **samouczeniem** oraz **samoorganizacją?**
2. Sieci Kohonena są często nazywane narzędziami, pozwalającymi „zaglądać w głąb wielowymiarowych przestrzeni danych". Czy potrafisz uzasadnić takie określenie?
3. Jednym z zastosowań sieci Kohonena jest użycie ich jako „detektora nowości". Sieć w takim zastosowaniu ma zasygnalizować fakt, że oto pojawił się zestaw sygnałów wejściowych, który nigdy wcześniej nie występował (ani w dokładnie takiej formie, ani w żadnej podobnej). Automatyczne sygnalizowanie takiej sytuacji może mieć istotne znaczenie na przykład przy wykrywaniu kradzieży karty kredytowej albo telefonu komórkowego – złodziej z reguły posługuje się nimi inaczej, niż czynił to wcześniej właściciel. Jak sądzisz – w jaki sposób sieć Kohonena sygnalizuje, że spotkała się z sygnałem mającym znamiona „nowości"?
4. Zbadaj przebieg procesu samoorganizacji po zmianie kształtu podobszaru przestrzeni sygnałów wejściowych, z którego rekrutowane są losowo pokazywane sygnały wejściowe i który jest wypełniany „siatką" rozpinaną przez sieć. Program pozwala Ci wybrać, czy wypełniany obszar jest kwadratem, czy trójkątem, czy też ma kształt krzyża. Zbadaj skutki tego wyboru. Jeśli jesteś zaawansowany w programowaniu, to możesz także podłubać trochę w programie i dodać jakieś dalsze kształty obszarów od siebie.
5. Zbadaj, jaki wpływ na proces samoorganizacji ma współczynnik uczenia dla neuronu – zwycięzcy, nazwany w programie **Alpha0**. Zmiany tego współczynnika powodują przyspieszenie uczenia (gdy współczynnik jest zwiększany) lub „uspakajanie" procesu, gdy współczynnik jest zmniejszany.

Współczynnik ten jest z kroku na krok automatycznie zmniejszany (patrz niżej), w wyniku czego proces uczenia, na początku szybki i dynamiczny, staje się w miarę upływu czasu coraz bardziej stabilny. Możesz jednak także i to zmienić, a potem przeanalizować zaobserwowane efekty.

6. Zbadaj, jaki wpływ na proces samoorganizacji mają zmiany współczynnika uczenia dla neuronów wchodzących w skład sąsiedztwa zwycięskiego neuronu. Współczynnik ten nazwany jest w programie **Alpha1.** Im ten współczynnik jest większy, tym efekt „ciągnięcia" za zwycięskim neuronem jego sąsiadów jest bardziej zauważalny. Zbadaj i opisz skutki zmian wartości tego współczynnika (a także stosunku **Alpha0/Alpha1**) oraz ich wpływ na zachowanie sieci.

7. Zbadaj, jak na zachowanie i proces samoorganizacji sieci wpływa zmieniany **zasięg sąsiedztwa**. Liczba ta określa, jak wiele neuronów wchodzi w skład sąsiedztwa, czyli jak wiele neuronów podlega wymuszonemu uczeniu przy samouczeniu się „zwycięzcy". Liczba ta powinna być zależna od rozmiarów sieci i tak właśnie jest ustawiana automatycznie w podanym programie, jednak proponuję Ci w ramach ćwiczeń dokładne przebadanie jej wpływu na zachowanie sieci. Warto przy tych eksperymentach zauważyć, że większe wartości zasięgu sąsiedztwa dość mocno spowalniają proces uczenia.

8. Zbadaj, jak na zachowanie i proces samoorganizacji sieci wpływa zmienianie współczynnika **EpsAlpha**, sterującego procesem zmniejszania (z kroku na krok) współczynników uczenia **Alpha0** i **Alpha1.** Zauważ, że im współczynnik ten będzie mniejszy, tym bardziej radykalnie zmniejszać się będą współczynniki uczenia i tym szybciej stabilizować się będzie proces uczenia. Możesz – jeśli zechcesz – dać ten współczynnik równy **1** i wtedy współczynniki nie będą się zmieniały w trakcie uczenia, albo nawet dać tu wartość nieco większą od 1, co spowoduje w miarę upływu czasu coraz bardziej brutalne uczenie sieci (oj, co to się będzie działo!).

9. Zbadaj, jak na zachowanie i proces samoorganizacji sieci wpływa zmienianie zasięgu sąsiedztwa. Możesz zmieniać współczynnik **EpsNeighb** który steruje procesem zwężania (z kroku na krok) zasięgu sąsiedztwa. Spróbuj porównać skutki zmieniania tego współczynnika z procesami, jakie zaobserwowałeś w odniesieniu do współczynnika **EpsAlpha**.

10. Zbadaj, jakie efekty dają próby „przeuczania" sieci, która początkowo nauczy się wypełniać aktywny podobszar przestrzeni sygnałów wejściowych przypominający – na przykład – krzyż, a potem może się rozwinąć do prostokąta lub zwinąć do trójkąta. Pamiętaj, że przy eksperymentach z „przeuczaniem" sieci trzeba po zmianie kształtu obszaru powiększyć współczynnik **Alpha0**, a także **Alpha1** i zasięg sąsiedztwa.

11. **Zadanie dla zaawansowanych:** Zmodyfikuj program w taki sposób, żeby możliwe było w nim modelowanie zadań, w których sieć Kohonena styka się ze zjawiskiem **silnie nierównomiernego** prawdopodobieństwa pojawiania się punktów pochodzących z różnych regionów przestrzeni sygnałów wejściowych. Przeprowadź proces samouczenia i sprawdź, że w nauczonej sieci znacznie więcej neuronów wyspecjalizuje się w rozpoznawaniu sygnałów pochodzących z regionów przestrzeni **częściej** reprezentowanych w zbiorze uczącym. Porównaj ten efekt ze strukturą mapy kory mózgowej, przedstawioną na rysunku 10.40. Jakie wnioski możesz wyciągnąć z tego doświadczenia?

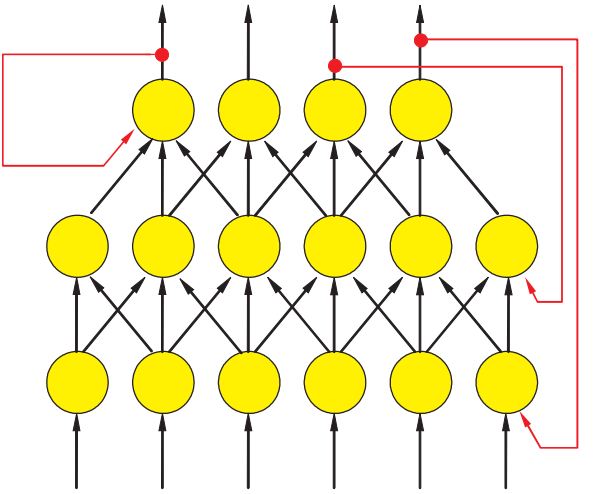
12. **Zadanie dla zaawansowanych:** Zbuduj program symulujący zachowanie robota, opisanego w podrozdziale 10.9. Przeprowadź próby, symulujące gromadzenie umiejętności kojarzenia bodźców sensorycznych, opisujących stan środowiska, z korzystnymi dla robota zachowaniami (powodującymi korzystne zmiany w jego symulowanym otoczeniu). Zbadaj, jaka reprezentacja wiedzy o otoczeniu (symulowanym środowisku robota) zostanie pozyskana i wykorzystana przez samoorganizującą się sieć neuronową, będącą jego „mózgiem". Zmieniając właściwości środowiska (które Ty symulujesz, więc możesz narzucić dowolne obowiązujące w nim prawa), zaobserwuj, które z tych właściwości robot potrafi „odkryć" w swojej samoorganizującej się sieci neuronowej, a które okażą się za trudne.

# 11. Sieci rekurencyjne

## 11.1. Co to jest sieć neuronowa rekurencyjna?

Dzięki przykładom pokazanym w poprzednich rozdziałach wiesz już, jak odbywa się uczenie sieci neuronowej – jedno lub wielowarstwowej, liniowej lub nieliniowej, uczonej przez nauczyciela lub zdobywającej wiedzę „na własną rękę", krótko mówiąc – **prawie** dowolnej.

Prawie, gdyż nie rozważałeś jeszcze sieci **rekurencyjnych**, czyli sieci zawierających **sprzężenia zwrotne.** Sprzężenia zwrotne są to takie połączenia, które zawracają sygnały z dalszych (od wejścia) neuronów sieci do neuronów warstwy wejściowej lub do wcześniejszych warstw ukrytych (rys. 11.1). Wbrew pozorom jest to bardzo istotna i znacząca innowacja. Jak się za chwilę przekonasz, taka sieć, w której występuje sprzężenie zwrotne, ma istotnie bogatsze właściwości i możliwości obliczeniowe od klasycznej sieci, w której dozwolony jest wyłącznie jednokierunkowy przepływ sygnałów – od wejścia do wyjścia.

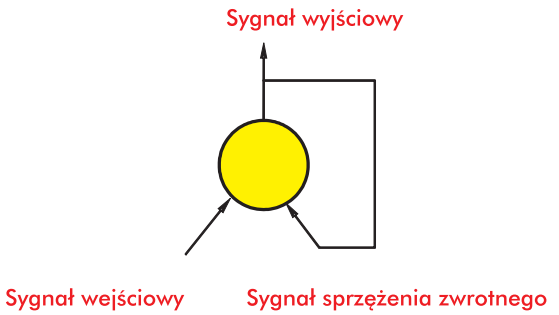


Rys. 11.1. Przykładowa struktura sieci rekurencyjnej. Wyróżniono czerwonym kolorem połączenia będące sprzężeniem zwrotnym

W sieciach ze sprzężeniami zwrotnymi występować będą zjawiska i procesy, których na próżno szukałbyś w sieciach jednokierunkowych. Jednorazowo pobudzona sieć ze sprzężeniem zwrotnym może generować całą sekwencję sygnałów i zjawisk, ponieważ sygnały z wyjścia (będące wynikiem przetwarzania sygnałów w pewnym *n*-tym kroku), trafiając z powrotem na wejścia neuronów, powodują wytworzenie nowych, z reguły zupełnie innych sygnałów w (n+1) kroku. Z takich powikłanych, „zapętlonych" obiegów sygnałów biorą się w sieciach rekurencyjnych specyficzne zjawiska i procesy – na przykład gasnące albo narastające oscylacje, gwałtowne narastania lub równie gwałtowne dławienia sygnałów, czy wreszcie tajemnicze, chaotyczne błądzenia (sprawiające wrażenie przebiegów całkowicie nie zdeterminowanych).

Takie zapętlone sieci, zawierające sprzężenia zwrotne, są jednak mało popularne, ponieważ są dość trudne do analizy. Trudności te wynikają z faktu, że sygnały mogą w nich długo krążyć w obwodach zamkniętych (od wejścia do wyjścia i z powrotem na wejście – co znowu zmienia wyjście itd.), w wyniku czego sieć neuronowa odpowiadając na każdy, nawet podany przez krótką chwilę sygnał wejściowy, przechodzi przez długą sekwencję rozmaitych stanów pośrednich, zanim ustalą się w niej wszystkie potrzebne sygnały.

Rozważ na prostym przykładzie, jak to się dzieje. Wyobraź sobie sieć złożoną z jednego tylko neuronu (liniowego, żeby było prościej). Ma on dwa wejścia – za pomocą jednego wprowadzisz do neuronu sygnał wejściowy, a na drugie wejście podasz sygnał z wyjścia tworząc w ten sposób sprzężenie zwrotne (rys. 11.2). Czy może być coś prostszego?

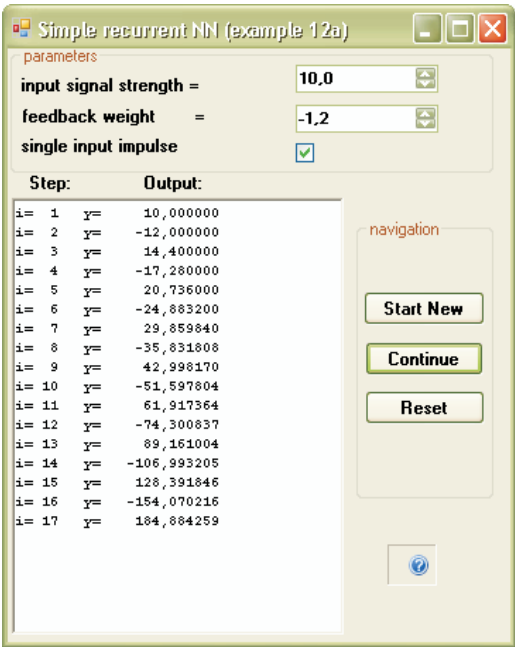



Rys. 11.2. Najprostsza struktura neuronowa ze sprzężeniem zwrotnym

Wypróbuj teraz działanie tej „sieci". Jak zwykle, proponuję Ci do tego celu prosty program **Example12a**, w którym będziesz mógł sam ustalić parametry rozważanej sieci (współczynnik wagowy (**feedback weight**), za pośrednictwem którego wprowadzany będzie do neuronu sygnał sprzężenia zwrotnego), a także sygnał wejściowy (**input signal strength**), od którego sieć wystartuje. Dodatkowo możesz określić (uaktywniając albo nie pole **sin-**

**gle_input_impulse**), czy sygnał wejściowy ma być stale podawany na wejściu sieci, czy też ma się pojawiać chwilowo (jednorazowo) wyłącznie na początku symulacji. Program ten będzie z kroku na krok obliczał sygnały krążące w sieci, pokazując jej zachowanie. Przy jego pomocy zauważysz łatwo kilka prawidłowości:

⇒ Opisana sieć istotnie wykazuje złożone formy dynamiki: po jednorazowym (impulsowym) podaniu sygnału na wejściu sieci – na jej wyjściu zapoczątkowany zostaje długotrwały proces, podczas którego sygnał wyjściowy zmienia się wielokrotnie, zanim osiągnie stan równowagi – jeśli go w ogóle osiągnie (rys. 11.3).

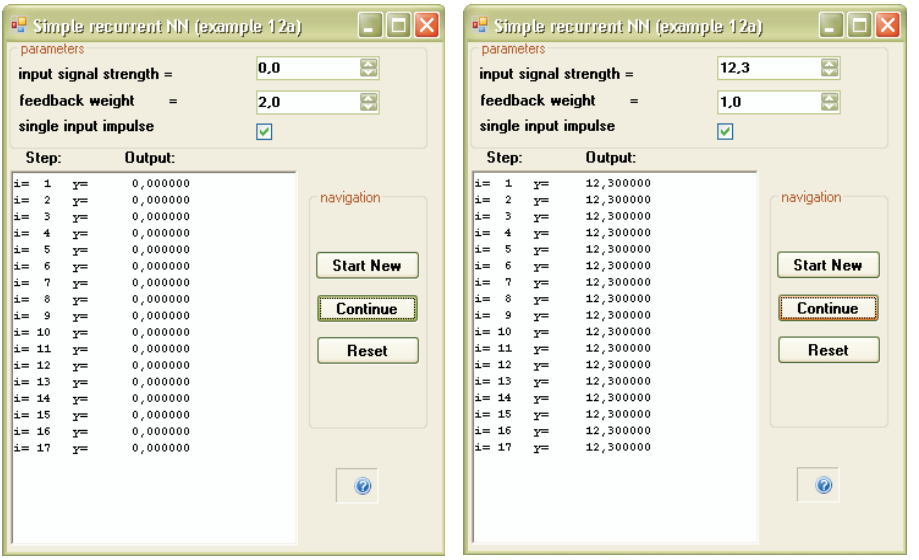



Rys. 11.3. Przykładowy wynik działania programu **Example 12a** ujawnia złożone działanie systemu ze sprzężeniem zwrotnym nawet w bardzo prostej sytuacji wejściowej

⇒ Równowaga w tej prostej sieci może być osiągnięta (bez trwającego w trakcie całej symulacji sygnału wejściowego) jedynie w taki sposób, że stale iloczyn pewnego sygnału wyjściowego po przemnożeniu przez wagę odpowiedniego wejścia, dawać będzie sygnał wyjściowy równy dokładnie sygnałowi sprzężenia zwrotnego, potrzebnemu do tego, by wytworzyć dany sygnał wyjściowy (sygnały na obu wejściach neuronu będą się równoważyły wzajemnie)**.**

⇒ Sygnał wyjściowy, przy którym spełniony jest warunek opisany wyżej, nazywany jest **atraktorem**. Nazwa ta będzie potem jeszcze obszerniej wyjaśniona.

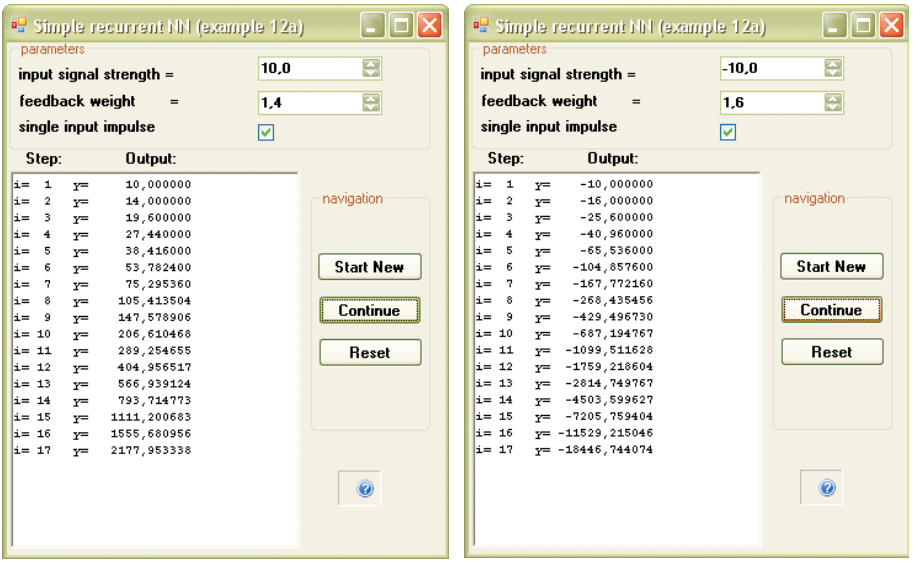
⇒ Położenie atraktora zależne jest od parametrów sieci. Dla sieci o współczynniku wagowym sprzężenia zwrotnego mającym wartość **1 każdy** punkt jest atraktorem, natomiast dla dowolnej innej sieci stan równowagi daje się uzyskać dopiero wtedy, gdy sygnał wyjściowy ma wartość równą zero (rys. 11.4). Jest to cecha rozważanej tutaj prostej sieci liniowej, w sieciach nieliniowych atraktorów może być więcej, z czego będziemy skwapliwie korzystali.



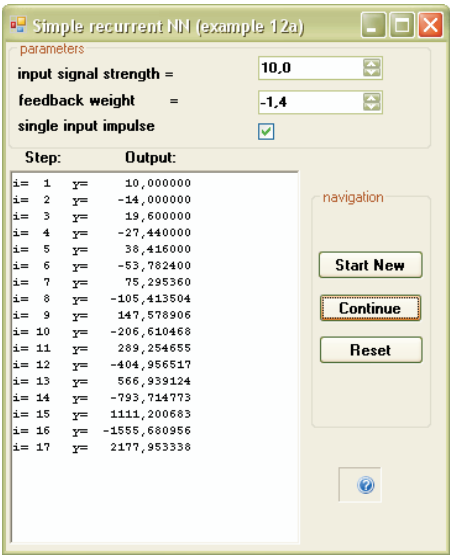

Rys. 11.4. Dwa sposoby uzyskania stanu równowagi w sieci liniowej: zerowe sygnały (po lewej) albo jednostkowe wzmocnienie sprzężenia zwrotnego przy równoczesnym braku sygnału wejściowego (po prawej)

⇒ Jeśli wartość współczynnika wagi synaptycznej w obwodzie sprzężenia zwrotnego jest **dodatnia** (tworzy się w ten sposób tak zwane dodatnie sprzężenie zwrotne) – przebiegi w systemie mają charakter **aperiodyczny**, to znaczy nie wykazują oscylacji – rys. 11.5. Warto zauważyć, że w systemie takim mogą występować procesy przebiegające wyłącznie w obrębie dodatnich wartości sygnałów (na lewym rysunku) albo wyłącznie w obszarze wartości ujemnych (na prawej części rysunku), przy czym typowe wartości dodatnie stają się w miarę upływu czasu coraz bardziej dodatnie, a ujemne – coraz bardziej ujemne. Natomiast wcale nie występują procesy, w których sygnały zmieniałyby znak.

⇒ Jeśli wartość współczynnika wagi synaptycznej w obwodzie sprzężenia zwrotnego jest **ujemna** (w wyniku czego powstaje tak zwane ujemne




Rys. 11.5. Typowe przebiegi w systemie ze sprzężeniem zwrotnym dodatnim




Rys. 11.6. Typowy przebieg w systemie ze sprzężeniem zwrotnym ujemnym

sprzężenie zwrotne) – przebiegi w systemie mają charakter **periodyczny**, to znaczy wykazują oscylacje – rys. 11.6.

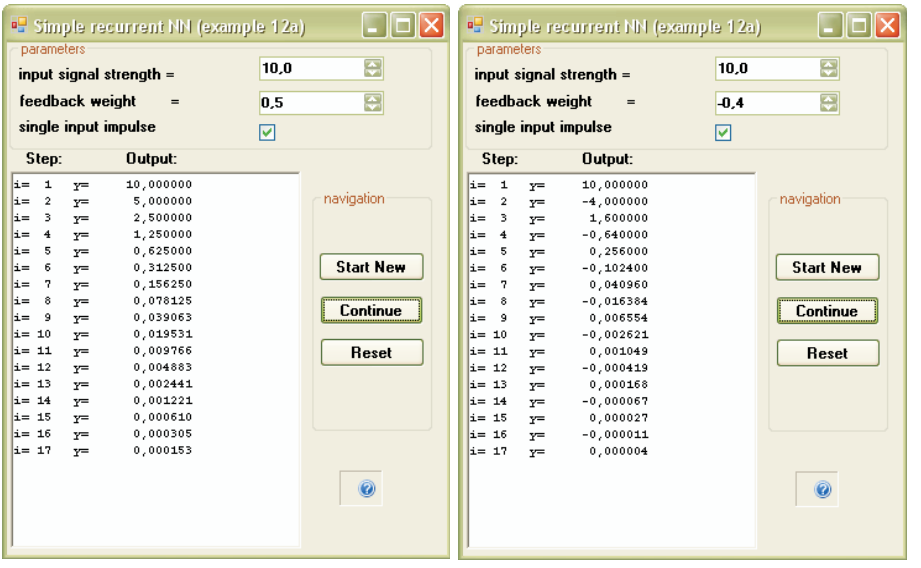
⇒ Charakter oscylacji w systemie ze sprzężeniem zwrotnym ujemnym zależy od parametrów sieci. Czasem drgania gwałtownie narastają, sieć miota

się od dużych wartości ujemnych do jeszcze większych wartości dodatnich, aby potem wygenerować jeszcze większy sygnał ujemny itd. Taka targana konwulsjami sieć musi doprowadzić do katastrofy, bo jej zachowanie przypomina działanie mózgu człowieka chorego na epilepsję albo zachowanie startującej rakiety kosmicznej, która utraciła stabilność i za chwilę się rozbije. Czasem jednak drgania spokojnie zanikają do zera, sieć się uspokaja i stabilizuje – tak właśnie działają wszystkie dobrze zaprojektowane systemy automatyki.

⇒ Obserwując zachowanie systemów ze sprzężeniem zwrotnym zauważysz, jak bardzo są one wrażliwe na wartości występujących w nich parametrów. Podobnego zjawiska nie wykazują systemy pozbawione sprzężenia zwrotnego, a więc jest to cecha wyróżniająca rekurencyjne sieci neuronowe (i inne rekurencyjne systemy).

⇒ Przeanalizujmy opisane wyżej zjawisko od strony ilościowej. Wykonując kilka próbnych symulacji, łatwo się przekonasz, że jeśli **bezwzględna wartość** współczynnika wagi w sprzężeniu zwrotnym jest większa od pewnej ustalonej wartości, określanej jako **granica stabilności**, wówczas zarówno w systemie ze sprzężeniem zwrotnym ujemnym, jak i w systemie ze sprzężeniem zwrotnym dodatnim występują sygnały, których bezwzględne wartości bezustannie wzrastają (rys. 11.5 i 11.6). Zjawisko takie znane jest jako niestabilne zachowanie sieci.

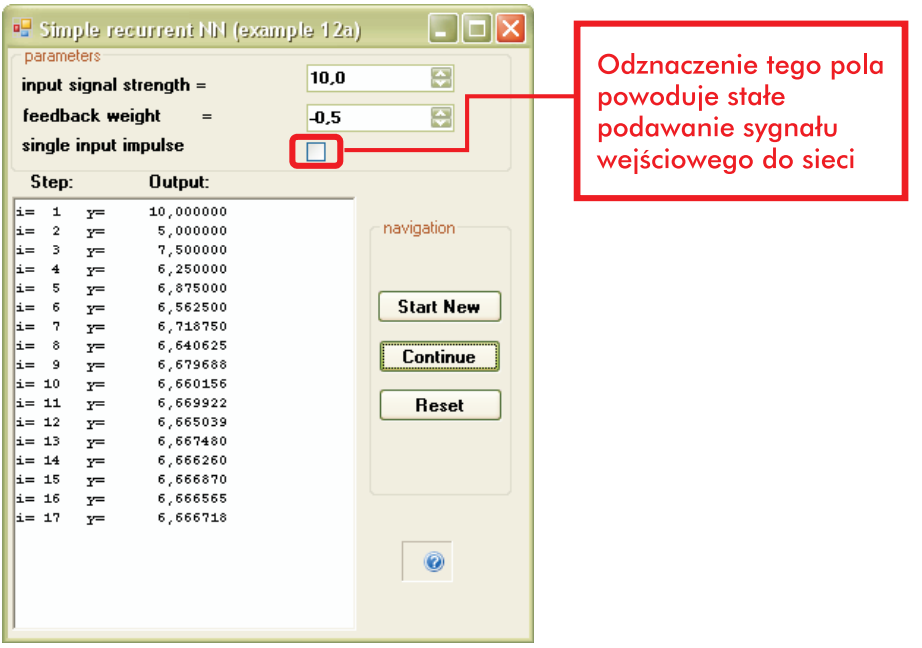
⇒ Jeśli jednak bezwzględna wartość współczynnika wagi w sprzężeniu zwrotnym jest mniejsza od tej ustalonej wartości – wówczas zarówno układ




Rys. 11.7. Stabilne przebiegi w systemie ze sprzężeniem zwrotnym po zmniejszeniu wartości współczynnika wagi znacznie poniżej granicy stabilności

ze sprzężeniem zwrotnym dodatnim, jak i układ ze sprzężeniem zwrotnym ujemnym dąży do stanu równowagi (rys. 11.7).

⇒ Sygnał wejściowy może także być podawany do sieci przez cały czas trwania symulacji (program daje taką możliwość, wystarczy usunięcie zaznaczenia pola **single_input_impulse**). W takim przypadku równowaga też może być osiągana, ale wartość sygnału wyjściowego, dla którego proces się stabilizuje jest wtedy inna i oczywiście zależna od wartości podanego sygnału wejściowego.




Rys. 11.8. Zachowanie systemu ze sprzężeniem zwrotnym przy stałym podawaniu do niego ustalonej wartości sygnału wejściowego

## 11.2. Jakie właściwości mają sieci ze sprzężeniem zwrotnym?

Podsumujmy wyniki niektórych przeprowadzonych wyżej badań, wyciągając z nich te wnioski, które przydadzą Ci się w przyszłości.

Przebieg sygnałów wejściowych w sieci ze sprzężeniem zwrotnym może wykazywać dwojakiego rodzaju zmienność: jeśli współczynnik wagi sprzężenia zwrotnego jest dodatni (tzw. **dodatnie sprzężenie zwrotne**), to sygnał zmienia się jednokierunkowo (**aperiodycznie**), natomiast przy ujemnym współczynniku wagi sprzężenia zwrotnego (sprzężenie zwrotne ujemne, zwa-

ne też **regulacyjnym**) pojawiają się **oscylacje** (sygnał na wyjściu przyjmuje na przemian wartości mniejsze i większe, często wręcz dodatnie i ujemne). Gdyby neuron był nieliniowy możliwa byłaby jeszcze jedna forma zachowania systemu, a mianowicie **chaotyczne błądzenie** sygnałów ze wszystkimi cudeńkami towarzyszącymi współczesnej teorii chaosu („efekt motyla", dziwne atraktory, fraktale, zbiory Mandelbrota itd.) – ale to już jest temat na osobne opowiadanie. Jeśli jednak jesteś ciekaw, co się za tym kryje (a są to fascynujące rzeczy!) – to wpisz któreś z podanych haseł do wyszukiwarki Google i obejrzyj, co Ci zaprezentuje. Naprawdę warto!

Niezależnie od podziału form zachowania sieci na aperiodyczne i periodyczne zauważyłeś pewnie, że w działaniach sieci ze sprzężeniem zwrotnym dają się wyróżnić zachowania **stabilne** (sygnały mają ograniczone wartości i zwykle po kilku krokach „zbiegają się" do jakiejś ustalonej końcowej wartości) oraz zachowania **niestabilne** – kolejne wartości sygnału wyjściowego są coraz większe (co do bezwzględnej wartości) i w końcu „uciekają" poza zakres wartości dopuszczalnych. W symulacyjnym programie komputerowym pojawia się wtedy błąd matematyczny określany jako „nadmiar zmiennoprzecinkowy", ale w systemie zbudowanym jako układ elektroniczny albo mechaniczny w takim przypadku niezawodnie coś się spali albo wybuchnie!

Może zainteresuje Cię fakt, że w rzeczywistych sieciach neuronowych takie zjawiska też na co dzień zachodzą, tylko przypuszczalnie nigdy nie patrzyłeś na nie z tego punktu widzenia.

Dodatnie sprzężenie zwrotne polega na tym, że im więcej myślisz o jakiejś sprawie, tym bardziej Cię ona fascynuje, więc tym bardziej się nią zajmujesz, aż wreszcie zaprząta Cię ona bez reszty. Spotykałeś się na pewno z taką sytuacją, prawda? A przecież jest to typowy efekt wzmacniania dodatnich wartości sygnałów w systemie z dodatnim sprzężeniem zwrotnym, dokładnie taki, jak pokazałem Ci po lewej stronie rysunku 11.5.

Podobnie zaobserwowałeś także efekt odwrotny: jeśli masz negatywny stosunek do jakiejś sprawy (na przykład do jakiejś formy rozrywki albo do jakiegoś przedmiotu szkolnego...), to kolejne doświadczenia utwierdzają Cię tylko w przekonaniu, że jest to głupie, nieciekawe i absolutnie nie dla Ciebie. Zauważ podobieństwo tej sytuacji do przebiegu, który pokazałem Ci po prawej stronie rysunku 11.5. Jeśli taki zamknięty obieg dodatniego sprzężenia zwrotnego nie zostanie przerwany – to może dojść nawet do zaburzeń psychicznych. Jest to szczególnie niebezpieczne wtedy, gdy w ten sposób utrwalasz i wzmacniasz swój stosunek do jakiejś osoby. W przypadku wzmacniania wartości ujemnych nazywa się to psychozą maniakalną, a w przypadku wartości dodatnich – stanem ostrego zakochania i z niewiadomych powodów nie jest to leczone ☺.

Zauważ, że w rzeczywistym systemie nerwowym spotyka się także (na szczęście rzadko) zjawisko niestabilności. Neurolodzy nazywają to napadem padaczkowym, a chorobę, przy której zjawisko to występuje, epilepsją. Epilepsja jest groźną chorobą, objawiającą się gwałtownymi skokami wartości sygnałów w mózgu (można je wykryć rejestrując na powierzchni czaszki sygnał elektryczny wywoływany przez pracujący mózg, tzw. zapis EEG) i wywołanymi przez nie gwałtownymi skurczami mięśni, powodującymi nie kontrolowane i bardzo gwałtowne konwulsje całego ciała. Choroba ta budziła w przeszłości duży lęk albo traktowana była jako przejaw nawiedzenia człowieka przez złe lub dobre duchy („taniec Świętego Wita"). Podobno cierpiał na nią sam Juliusz Cezar, możesz więc od jutra opowiadać swoim znajomym, że właśnie symulujesz na swoim komputerze mózg Juliusza Cezara, co powinno ich wprawiać w stosowne osłupienie (współczynniki wag synaptycznych równe zero).

Proponuję, abyś sam spróbował wymyślić albo wykryć za pomocą eksperymentów wykonywanych na komputerze, jaka jest granica stabilności dla rozważanego tu prostego systemu ze sprzężeniem zwrotnym? Spróbuj sam odkryć, jakie czynniki decydują o tym, czy sieć zachowuje się stabilnie, czy niestabilnie. Ponieważ w następnym zdaniu będę to komentował – proponuję, żebyś tu na chwilę przerwał czytanie i poeksperymentował z programem **Example12a**, a potem porównasz to, co sam odkryłeś, z tym, co na ten temat ma do powiedzenia osiwiała od mądrości TEORIA.

Prowadząc badania z użyciem programu, zauważyłeś na pewno, że zmiany sygnału na wyjściu sieci zależą głównie od wartości występujących w niej współczynników; natomiast sygnał wejściowy ma na to znacznie słabszy wpływ – nawet jeśli spowodujesz, że sygnał wejściowy będzie podawany przez cały czas, a nie tylko w pierwszej chwili po wystartowaniu symulacji (odznaczając w programie pole opisane jako **single_input_impulse)**. Badając zachowanie sieci (lub zastanawiając się nad zastosowanym algorytmem), łatwo ustalisz, że naprawdę ważne jest to, czy współczynnik wagi dla sygnału sprzężenia zwrotnego jest – **na bezwzględną wartość** – mniejszy czy większy od **1**. Dla współczynników mniejszych od **1** masz stale do czynienia z procesem stabilnym – aperiodycznym przy wartościach dodatnich i oscylacyjnym przy wartościach ujemnych. Dla wartości współczynników większych od **1** przebieg jest zawsze niestabilny. Natomiast przy wartości współczynnika wynoszącej dokładnie **1** masz do czynienia przy dodatnim sprzężeniu zwrotnym z dziwną sytuacją, polegającą na tym, że każdy sygnał wejściowy okazuje się atraktorem sieci, a przy ujemnym sprzężeniu zwrotnym masz do czynienia ze stałymi, nie rosnącymi, ale i nie gasnącymi oscylacjami. Stan taki nazywany jest **granicą stabilności**.

W bardziej złożonych sieciach warunki stabilności są bardziej skomplikowane i dla wyznaczenia granicy stabilności używać trzeba bardzo zaawansowanych metod matematycznych (wyznaczniki Hurwitza, diagramy Nyquista, twierdzenie Lapunowa itp.), teoria sieci neuronowych ze sprzężeniem zwrotnym jest więc od lat bardzo wdzięcznym polem działania dla wszelkiego autoramentu teoretyków zajmujących się dynamiką złożonych systemów.

## 11.3. Po co komu takie sieci neuronowe „z pętelkami"?

Sytuacja w sieci złożonej z jednego neuronu liniowego jest wygodnie prosta i łatwa do przewidzenia, więc daje stosunkowo mało okazji do ciekawszych praktycznych zastosowań. Dlatego z ćwiczonego wyżej programu **Example12a** nie da się wycisnąć żadnych wyników, które by się kojarzyły z jakimiś użytecznymi efektami. Dopiero większe sieci ze sprzężeniem zwrotnym, obejmujące kilkanaście czy kilkadziesiąt neuronów (zwłaszcza nieliniowych!) mogą wykazywać naprawdę ciekawe i naprawdę złożone zachowania – co jest kluczem do ich praktycznych zastosowań. Warto jednak wiedzieć, że w takich sieciach o większej liczbie neuronów wzajemnie przekazujących sobie swoje sygnały wyjściowe przez sprzężenia zwrotne możliwe są znacznie bardziej złożone sytuacje od stabilnych i niestabilnych zachowań prostego pojedynczego neuronu, wyżej przeze mnie w zarysie opisanych i przez Ciebie (mam nadzieję!) własnoręcznie dogłębnie przebadanych. W takiej złożonej sieci stany równowagi mogą być osiągane dla różnych zestawów wartości sygnałów wyjściowych (niekoniecznie mających mało ciekawą wartość 0), przy czym możliwe jest takie dobranie struktury połączeń i parametrów sieci, że te **osiągane przez sieć stany równowagi odpowiadać będą rozwiązaniom pewnych zadań.**

Jest to klucz do większości niebanalnych zastosowań tych sieci. Przykładowo wymienić tu można sieci, w których stan równowagi odpowiada rozwiązaniu pewnego **problemu optymalizacyjnego** (poszukiwania najkorzystniejszych decyzji, zapewniających maksymalny zysk lub minimalne straty, przy uwzględnieniu ograniczeń – na przykład ograniczonej mocy sygnałów sterujących lub ograniczonych środków na inwestycje). Znane są sieci tego typu rozwiązujące tą metodą słynne zadanie komiwojażera (pisałem o tym dość dokładnie w mojej książce *Sieci neuronowe*, która jest w całości dostępna w Internecie), podejmowano z pomocą sieci neuronowych zadania optymalnego dzielenia ograniczonych zasobów (na przykład wody), były podejmowane próby (niektóre bardzo udane!) wykorzystania sieci do optymalnego doboru struktury tzw. portfela akcji podczas gry na giełdzie itp. Robili to moi współpracownicy, badali to moi magistranci i doktoranci, pewne badania pro-

wadziłem też osobiście. Nie o tym jednak będę dalej pisał, bo wolę, żebyś doskonalił swój umysł, poznając tajniki funkcjonowania sieci neuronowych, a nie hazardował się grą na giełdzie, gdzie nawet mając wspomaganie ze strony inteligentnych sieci neuronowych znacznie częściej się przegrywa, niż wygrywa.

Dlatego w dalszej części tego rozdziału pokażę Ci inny przykład zadania, w którym osiągnięcie przez sieć jednego z wielu możliwych stanów równowagi może być interpretowane jako rozwiązanie pewnego użytkowo ważnego informatycznego problemu – budowy tak zwanej **pamięci skojarzeniowej**.

Pamięci tego typu (skojarzeniowe, asocjacyjne, adresowane semantycznie, kontekstowe – w literaturze stosowane są różne nazwy) są od dawna marzeniem wszystkich informatyków, znużonych prymitywizmem aktualnie dostępnych metod wyszukiwania informacji w typowych bazach danych. Przyjrzyjmy się bliżej, o co w takich pamięciach właściwie chodzi.

Doskonale wiesz, że zgromadzenie w komputerze nawet milionów rekordów zawierających jakieś ważne informacje – nie przedstawia dziś żadnej trudności. Natomiast kłopoty zaczynają się, gdy trzeba do jednej z tych wiadomości szybko dotrzeć. Jeśli da się wskazać jakieś hasło, słowo kluczowe albo konkretną *wartość jakiegoś pola*, która odróżnia poszukiwaną wiadomość od wszystkich innych – sprawa jest prosta. Komputer odnajdzie i udostępni potrzebną informację, a jeśli w dodatku zadbano o odpowiednie przygotowanie (indeksowanie) bazy – zrobi to bardzo szybko i sprawnie. Zapewne robiłeś to wielokrotnie, wpisując rozmaite pytania do przeglądarki Google i otrzymując odpowiedzi (nie zawsze trafne…) z głębin Internetu.

Gorzej jest jednak wtedy, gdy nie znasz takiego słowa kluczowego czy innego elementu identyfikującego potrzebne Ci dane, natomiast masz pewien ogólny pogląd, na jaki temat potrzebna jest Ci określona informacja. Ponieważ nie wiesz, jak ta potrzebna Ci wiadomość ma dokładnie wyglądać – typowe, funkcjonujące w bazach danych albo w Internecie systemy wyszukiwania informacji okazują się zawodne i frustrująco mało skuteczne. Albo zasypują Cię mnóstwem niepotrzebnych wiadomości, albo nie są w stanie odnaleźć potrzebnej informacji, pomimo że dokładny przegląd bazy pokazuje, że jest ona dostępna i mogła być dostarczona. Z dwojga złego lepszy jest oczywiście scenariusz, kiedy dostajesz nadmiar danych, wśród których są oczywiście także te użyteczne, tylko jak je potem wydobyć? To dużo niewdzięcznej i niepotrzebnej pracy.

A przecież Twój umysł działa inaczej. Wystarcza Ci zwykle drobny ułamek wiadomości, by w mgnieniu oka odtworzyć całą informację. Czasem wystarczy jedno słowo, jakiś obraz, idea, wzór matematyczny – i już w umyśle kształtuje się kompletny zestaw wiadomości, odniesień, sugestii i wniosków.

Innym razem wystarczy ledwie uchwytny zapach, może fragment melodii, zachód słońca – aby uruchomić całą serię obrazów, wspomnień, nastrojów i uczuć...

Twoja pamięć potrafi więc odszukać wiadomość na podstawie drobnego jej fragmentu lub dotrzeć do niej wychodząc od innej wiadomości – w jakiś sposób z tą pierwszą powiązanej. Nasze komputery jeszcze tego nie potrafią. A czy potrafi to zrobić sieć neuronowa?

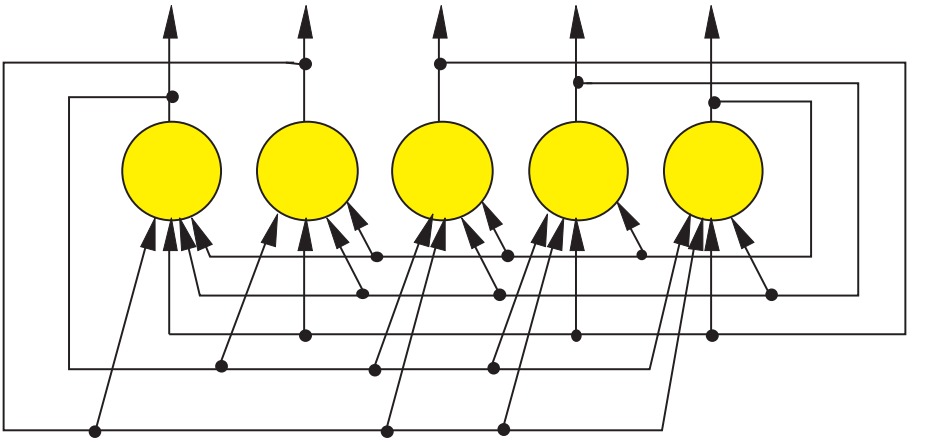
Spróbuj, a sam się przekonasz.

Zadanie to będzie niżej dokładniej omówione, więc szczegóły tej aplikacji poznasz już za kilka minut. Zanim to jednak nastąpi, muszę Ci o czymś powiedzieć. Zagadnienia pamięci skojarzeniowej są fragmentem obszerniejszej dziedziny wiedzy, zwanej **kognitywistyką** (ang. *cognitive science*). Budzi ona ostatnio rosnące zainteresowanie, zwłaszcza wśród filozofów, pedagogów i psychologów, ale pasjonuje się nią także wielu innych specjalistów, zwłaszcza fizyków, na co dzień zajmujących się teoretycznym opisem prostych układów fizycznych (na przykład cząstek elementarnych) lub specjalizujących się w globalnym opisywaniu systemu złożonego z bardzo wielu oddziaływujących na siebie cząstek (termodynamiką statystyczną). Szperając w bibliotece albo w Internecie, zauważysz, że zdumiewająco wielu fizyków opracowało i ogłosiło drukiem mnóstwo bardzo ambitnych matematycznie prac, opisujących zachowanie sieci ze sprzężeniami i procesy prowadzące do korzystniejszego ukierunkowania tego zachowania. Między innymi na tej zasadzie powstały tzw. **maszyny Boltzmanna** (wykorzystujące analogie procesów zachodzących w sieciach neuronowych ze zjawiskami termodynamicznymi, opisywanymi rozkładem Boltzmanna) i algorytmy **„symulowanego wyżarzania"**, o których jednak porozmawiamy innym razem. Kognitywistyką i symulacją sieci neuronowych zajmują się w Polsce między innymi tacy wybitni uczeni, jak prof. **Leszek Rutkowski** z Politechniki Częstochowskiej czy prof. **Józef Korbicz** z Uniwersytetu Zielonogórskiego. Obydwu mam zaszczyt zaliczać do swoich przyjaciół, chętnie więc bym napisał, że to właśnie oni wnieśli zasadniczy wkład do rozwoju tej teorii. Jednak stosując zasadę *„amicus Plato, sed magnis amica veritas"* muszę przyznać, że największy wkład do teorii sieci ze sprzężeniem zwrotnym wniósł inny fizyk, Amerykanin **John Hopfield**.

## 11.4. Jak jest zbudowana sieć Hopfielda?

Sieci Hopfielda są niewątpliwie najważniejszą i najczęściej stosowaną podklasą rekurencyjnych sieci neuronowych wykorzystywanych w praktyce, dlatego właśnie te sieci powinieneś teraz poznać. Charakteryzując je w sposób najbardziej ogólny, mogę stwierdzić, że sieci te stanowią krańcowe przeciwieństwo

sieci klasy *feedforward*, czyli sieci całkowicie pozbawionych sprzężeń zwrotnych, omawianych we wszystkich wcześniejszych rozdziałach. Sieci, w których **dopuszczono** sprzężenia zwrotne (tak zwane sieci rekurencyjne), mogą zawierać **pewną liczbę** sprzężeń zwrotnych, przy czym może ich być więcej albo mniej. Tymczasem w sieci Hopfielda sprzężenia zwrotne są po prostu regułą (rys. 11.9).



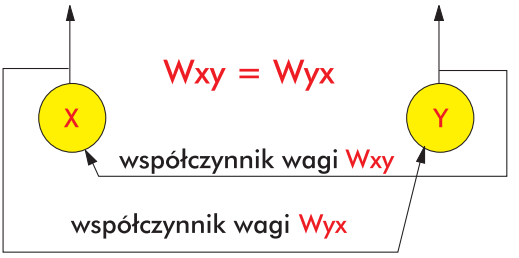
Rys. 11.9. Struktura prostej sieci Hopfielda

**Wszystkie** połączenia, które w tej sieci występują, są sprzężeniami zwrotnymi, **wszystkie** sygnały wyjściowe są wykorzystywane jako wejścia i **wszystkie** wejścia do **wszystkich** neuronów przenoszą sygnały sprzężenia zwrotnego. W sieci Hopfielda każdy neuron jest związany z każdym innym w obrębie całej sieci na zasadzie obustronnego sprzężenia zwrotnego, jest to więc sieć sprzęgnięta zwrotnie w sposób krańcowy i kompletny. Sieć Hopfielda stanowi więc całkowite przeciwieństwo wszystkich omawianych wcześniej sieci, które bez wyjątku były sieciami typu *feedforward*, i w nich jakiekolwiek sprzężenia zwrotne były po prostu zabronione.

Sieci Hopfielda są tak ważne głównie dlatego, że procesy w nich zachodzące są zawsze stabilne, można więc bezpiecznie stosować je do rozwiązywania różnych zadań bez obawy, że wszystko nagle wyleci w powietrze. Stabilność procesów w sieci Hopfielda osiągnięto dzięki zastosowaniu trzech prostych zabiegów:

⇒ wprowadzono bardzo regularną (i prostą do realizacji zarówno w formie programu komputerowego, jak i w postaci specjalizowanych układów elektronicznych lub optoelektronicznych) strukturę wewnętrzną sieci, polegającą na tym, że w obrębie całej sieci neurony są łączone na zasadzie „każdy z każdym”. Przypomnij sobie, że ta sama prosta (ale kosztowna!) reguła kształ-

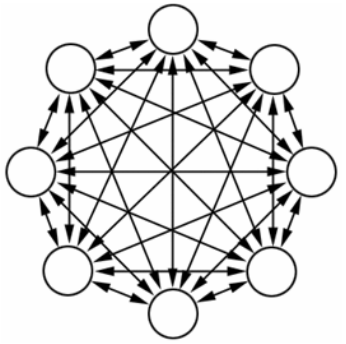
towania połączeń w sieci obowiązywała także w sieciach, które poznawałeś wcześniej – na przykład w sieci uczonej metodą *backpropagation* połączenia pomiędzy neuronami warstwy ukrytej i warstwy wyjściowej także były organizowane według zasady „każdy z każdym". Zatem sieć Hopfielda korzysta w tym zakresie z dobrze wypróbowanych wzorów – z tą tylko różnicą, że łączone ze sobą neurony nie stanowią oddzielnych warstw, tylko zarówno jako nadajniki informacji, jak i jako jej odbiorniki stanowią jedną i tę samą zbiorowość (patrz rys. 11.10);




Rys. 11.10. Schemat sieci Hopfielda podkreślający równoprawność wszystkich budujących ją neuronów oraz symetrię ich połączeń

⇒ zabroniono sprzężeń zwrotnych obejmujących pojedynczy neuron. Oznacza to, że sygnał wyjściowy z danego neuronu nie może być bezpośrednio podawany na jego wejście, co jak może zauważyłeś jest przestrzegane w strukturze sieci pokazanej na rysunku 11.9. Zauważ, że nie wyklucza to sytuacji, że sygnał wyjściowy z danego neuronu wpływa na swoją własną wartość w przyszłości, ponieważ możliwe jest sprzężenie zwrotne poprzez dodatkowe neurony pośredniczące, jednak wpływ tych dodatkowych „pośredników" jest zdecydowanie stabilizujący;

⇒ wprowadzane współczynniki wagowe muszą być symetryczne – to znaczy jeśli połączenie od neuronu o numerze $\boldsymbol{x}$ do neuronu o numerze $\boldsymbol{y}$ charakteryzuje się pewnym współczynnikiem wagi $\boldsymbol{w}$, to dokładnie taką samą wartość $\boldsymbol{w}$ ma współczynnik wagowy połączenia biegnącego od neuronu o numerze $\boldsymbol{y}$ do neuronu o numerze $\boldsymbol{x}$ (rys. 11.11).

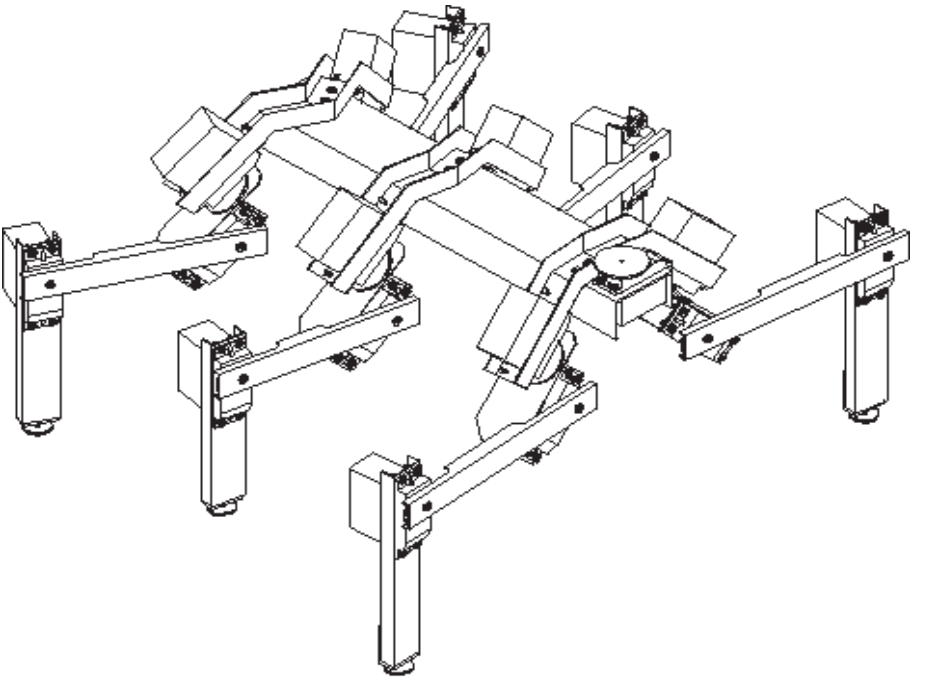


Rys. 11.11. Zasada symetrii obowiązująca w sieci Hopfielda

Wszystkie te warunki bardzo łatwo spełnić. Dwa pierwsze określają regularny i bardzo łatwy do uzyskania schemat połączeń sieci, a ostatni warunek jest automatycznie spełniony, jeśli do uczenia sieci wybierze się metodę **Donalda Hebba** (była ona niedawno omawiana w rozdziale 9, nie musimy jej więc teraz ponownie rozpatrywać).

Łatwość budowy i stosowania sieci Hopfielda powoduje, że są one bardzo popularne. Znajdują one liczne zastosowania – między innymi przy rozwiązywaniu zadań optymalizacji (o czym była wyżej mowa), a także przy generacji określonych sekwencji sygnałów, następujących po sobie w pewnej (modyfikowalnej) kolejności. Pozwala to za pomocą takich sieci tworzyć i wysyłać do różnych obiektów sygnały sterujące. Na tej zasadzie działają między innymi neuronowe systemy wytwarzające sygnały sterujące przestawianiem nóg w chętnie budowanych na całym świecie maszynach kroczących – dwunożnych, czworonożnych i sześcionożnych – patrz rys. 11.12.

„Mózg" sterujący ruchami takiego sztucznego wielonoga zawiera zawsze sieć posiadającą sprzężenia zwrotne, żeby mógł sam z siebie produkować cyklicznie zmieniające się sygnały sterujące. Zauważ, że niezależnie od liczby używanych nóg, kroczenie jest zawsze procesem cyklicznym, w którym



11.12. Przykład robota kroczącego, którego kończyny są sterowane zgodnie z cyklicznymi rytmami generowanymi przez „mózg" będący siecią Hopfielda

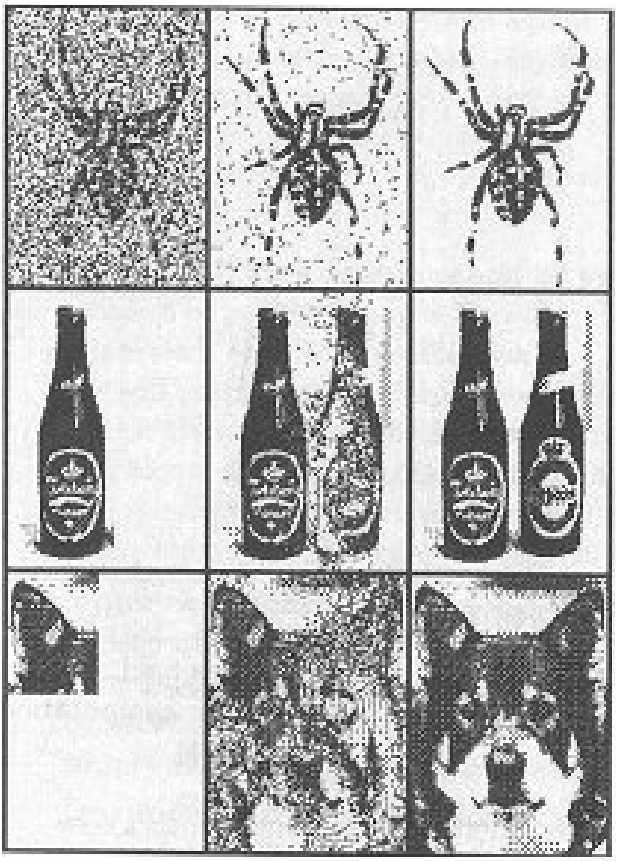
każda kończyna jest kolejno: podnoszona do góry, przenoszona do przodu, opuszczana aż do uzyskania stabilnego kontaktu z podłożem i przesuwana do tyłu w celu aktywnego przemieszczenia „ciała" kroczącego stworka do przodu, a potem przez pewien czas pełni tylko rolę podpory, podczas gdy napęd zapewniają pozostałe kończyny. Ponieważ kroczenie po nierównym podłożu wymaga nie tylko cyklicznego generowania wyżej opisanych ruchów, ale dodatkowo także musi uwzględniać mechanizm adaptacji do zmiennych sytuacji (na przykład gdy któraś noga wpadnie w szczelinę), przeto jako „mózg" kroczącego robota trzeba stosować jakieś narzędzie zdolne zarówno do generowania cyklicznych zachowań, jak i mające możliwość uczenia się oraz adaptacji do zmiennych sytuacji, czyli najczęściej właśnie sieć Hopfielda lub jakąś jej prostą modyfikację.

Takie uczące się kroczące automaty są bardzo ciekawe i będą miały w przyszłości duże zastosowanie przy eksploracji powierzchni odległych planet, wnętrza jaskiń albo dna oceanów, ale ten wątek chwilowo pominiemy. W kolejnym podrozdziale pokażę Ci bowiem, do czego sieć Hopfielda najlepiej się nadaje. Zbudujemy mianowicie razem pamięć asocjacyjną.

## 11.5. Jak działa sieć neuronowa jako pamięć skojarzeniowa?

Program **Example12b** zawiera model sieci Hopfielda, której zadaniem będzie zapamiętywanie i odtwarzanie prostych obrazów. Cały „smak" tej pamięci polega jednak na tym, że potrafi ona odtworzyć wiadomość (obraz) na podstawie sygnału silnie zniekształconego lub zakłóconego, działa ona zatem w sposób, który najczęściej nazywa się **autoasocjacją**[1]. Dzięki autoasocjacji sieć Hopfielda może automatycznie uzupełniać niekompletne dane. Na rysunku 11.13 pokazałem wielokrotnie przytaczany w różnych książkach i reprodukowany w Internecie (tu: pobrany ze stron ***www.cs.pomona.edu*** oraz dostępny także na stronie ***eduai.hacker.lt***) obraz pokazujący, jak sprawna potrafi być sieć Hopfielda przy usuwaniu zakłóceń z sygnału wejściowego oraz przy odtwarzaniu kompletnych danych w przypadku, kiedy dostarczone zostały jedynie ich fragmenty.

[1] **Autoasocjacja** oznacza taki sposób działania sieci, w którym określona wiadomość zostaje skojarzona sama z sobą. Dzięki autoasocjacji podanie nawet drobnego fragmentu zapamiętanej informacji powoduje, że informacja ta (na przykład obraz) zostaje przywołana w pamięci w całości i ze wszystkimi szczegółami. Alternatywą dla autoasocjacji jest **heteroasocjacja**, polegająca na tym, że jedna wiadomość (na przykład fotografia Babci) przywodzi wspomnienie innej informacji (na przykład smaku konfitur, które Babcia robiła). Sieci Hopfielda mogą pracować zarówno jako pamięci autoasocjacyjne albo jako heteroasocjacyjne, ale te pierwsze są prostsze i dlatego na nich się teraz skupimy.



Rys. 11.13. Przykłady działania sieci Hopfielda pracującej jako pamięć skojarzeniowa. Opis w tekście

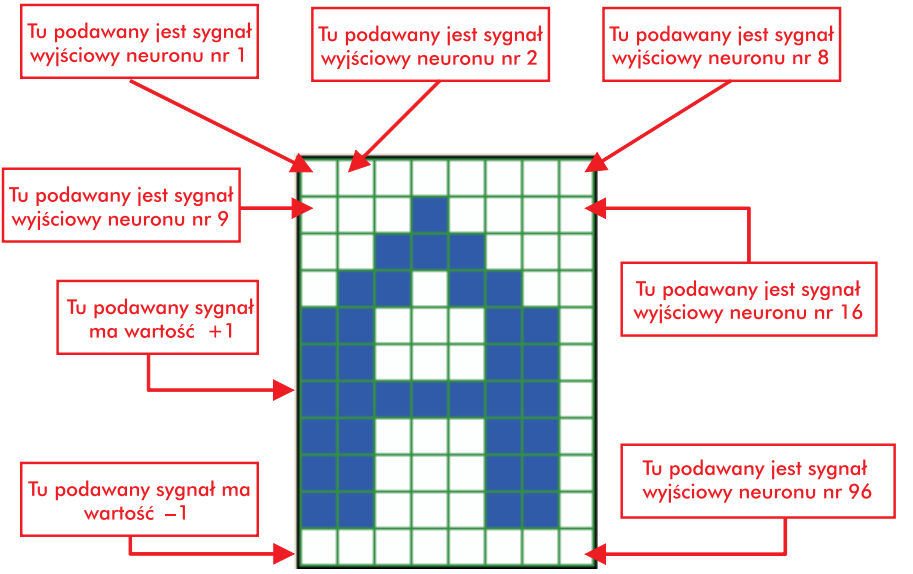
W trzech kolejnych wierszach tego rysunku pokazano każdorazowo po lewej stronie obraz, jaki został dostarczony jako dane wejściowe do sieci, w kolumnie środkowej pokazano stan przejściowy, gdy sieć poszukuje w swojej pamięci właściwego wzorca oraz po stronie prawej – efekt końcowy, to znaczy wynik „przypominania sobie" właściwego wzorca. Oczywiście, zanim uzyskano przedstawiane tu obrazy „przypominania sobie" przez sieć odpowiednich informacji (obrazów) – najpierw musiały one zostać utrwalone w niej drogą uczenia. Podczas uczenia pokazywano sieci wzorcowy obraz pająka, dwóch butelek oraz psiej mordki, a sieć zapamiętała sobie te wzorce i przygotowała się do ich przypominania. Jednak gdy już sieć została wytrenowana – to potrafiła robić naprawdę fajne rzeczy. Gdy pokazano jej bardzo silnie „zaszumiony" obraz pająka (górny wiersz rysunku 11.13) – to odtworzyła widok pająka bez zakłóceń. Gdy pokazano jej obraz jednej butelki – przypomniała sobie, że podczas uczenia prezentowano jej obraz dwóch butelek. Wreszcie wystarczyło jej pokazać samo ucho psa – żeby odtworzyła jego cały obraz.

Obrazki pokazane na rysunku 11.13 są ładne, ale ich odtwarzanie nie wiąże się z żadnym istotnym praktycznym zadaniem. Jednak pamięci autoasocjacyjne mogą być użyte do wielu użytecznych i praktycznych celów. Sieć na przykład może odtworzyć pełną sylwetkę nadlatującego samolotu w sytuacji, kiedy kamera zarejestrowała obraz niekompletny ze względu na to, że część zarysu przesłaniały obłoki. Ma to duże znaczenie w systemach obrony przeciwlotniczej, kiedy ważnym problemem jest szybkie rozstrzygnięcie problemu „swój – obcy". Sieć pracująca jako pamięć autoasocjacyjna może też uzupełnić niekompletne zapytanie kierowane do jakiegoś systemu informacyjnego typu baza danych. Jak wiadomo, z takiej bazy można uzyskać dużo pożytecznych informacji, pod warunkiem jednak, że się **poprawnie** zada pytanie. Jeśli pytanie do sieci będzie kierowane przez niewprawnego lub niestarannego użytkownika i nie odpowiada ustalonym schematom, to baza danych nie umie na nie odpowiedzieć. Pamięć autoasocjacyjna może pomóc w „dogadaniu się" z programem zarządzającym bazą danych, dzięki czemu wyszukiwanie potrzebnych informacji dokonywane będzie poprawnie nawet w przypadku nieprecyzyjnego (chociaż jednoznacznego) zapytania użytkownika. Sieć zdoła wtedy sama domyślić się wszystkich brakujących szczegółów, których roztrzepany (lub niedouczony) użytkownik nie podał – chociaż powinien. Autoasocjacyjna sieć Hopfielda pośredniczy wtedy między użytkownikiem a bazą danych, podobnie jak mądra bibliotekarka w szkolnej bibliotece, która potrafi zrozumieć, o jaką książkę pyta uczeń chcący wypożyczyć obowiązującą lekturę, dla której zapomniał autora, tytułu i wszelkich innych danych, ale wie o czym jest ta książka.

Sieć autoasocjacyjna może mieć liczne dalsze zastosowania, w szczególności potrafi usuwać zakłócenia i zniekształcenia różnych sygnałów – także wtedy, gdy stopień „zaszumienia" sygnału wejściowego wyklucza praktyczne użycie jakichkolwiek innych metod filtracji. Nadzwyczajna skuteczność sieci Hopfielda w takich przypadkach wynika z faktu, że sieć w istocie odtwarza wzorcową postać sygnału (najczęściej obrazu) ze swoich zasobów pamięciowych, zaś dostarczony zniekształcony wejściowy obraz ma ją tylko ukierunkować, „naprowadzić na trop" tego właściwego obrazka – pośród wszystkich obrazów, jakie mogą wchodzić w rachubę.

Myślę jednak, że dość już tej teorii i pora przejść do praktycznych ćwiczeń z użyciem programu **Example12b.** Ze względu na poglądowość i czytelność program prezentować Ci będzie działanie sieci Hopfielda na obrazkach, pamiętaj jednak, że absolutnie nie jest to jedyna możliwość – te sieci mogą podobnie zapamiętywać i odtwarzać **dowolne informacje,** pod warunkiem, że umówimy się co do tego, jak te informacje będziemy w sieci odwzorowywać i reprezentować.

Wyjaśnijmy więc najpierw związek modelowanej sieci Hopfielda z prezentowanymi przez program obrazkami. Każdy neuron sieci związujemy z jednym punktem (pikselem) obrazu. Jeśli na wyjściu neuronu jest sygnał **+1**, odpowiedni piksel jest czarny. Jeśli na wyjściu neuronu jest sygnał **–1**, odpowiedni piksel jest biały. Innych możliwości niż +1 i –1 nie przewidujemy, bo neurony budujące sieć Hopfielda są silnie nieliniowe i mogą znajdować się jedynie w tych dwóch wyróżnionych stanach (+1 albo –1), nie mogą natomiast przyjmować żadnych innych wartości. Rozważana sieć zawiera 96 neuronów, które – wyłącznie dla celów prezentacji wyników – uporządkowałem w formę matrycy o rozmiarach 12 wierszy i 8 pozycji w każdym wierszu. W związku z tym każdy konkretny stan sieci (rozumiany jako zbiór produkowanych przez sieć sygnałów wyjściowych) może być obserwowany jako jednobarwny obrazek o rozmiarach 12 × 8 pikseli, taki, jak pokazałem na rysunku 11.14.

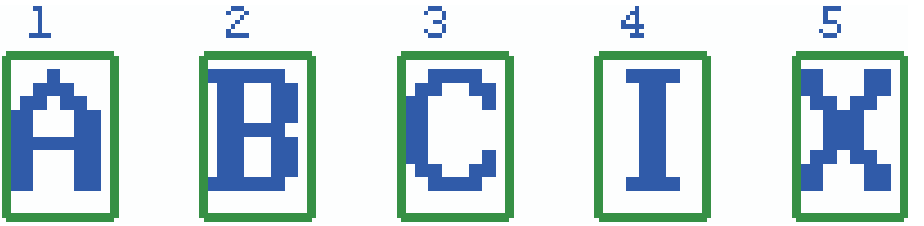



Rys. 11.14. Przykładowy obraz prezentujący, jaki jest rozkład wyjściowych sygnałów neuronów sieci

Rozważane obrazki mogłyby być całkiem dowolne, jednak dla wygody tworzenia zbioru zadań dla sieci zdecydowałem, że będą to obrazy liter (bo łatwo będzie je zadawać za pomocą klawiatury) albo całkowicie abstrakcyjne obrazki produkowane przez sam program według pewnych kryteriów matematycznych (o czym obszerniej napiszę dalej).

Program zapamięta pewną liczbę takich obrazków (podanych przez Ciebie lub wygenerowanych automatycznie) i sam (bez Twojego udziału!) wy-

świetli je jako wzorce do późniejszego odtwarzania (przypominania sobie). Na rysunku 11.15. możesz obejrzeć, jak taki przygotowany do zapamiętania zestaw wzorców może wyglądać. Oczywiście w zestawie zapamiętywanych przez sieć wzorców mogą się znaleźć dowolne inne litery lub cyfry, które możesz sam wygenerować z pomocą klawiatury, więc zestaw podany na rysunku 11.15 traktuj jako jeden z bardzo wielu możliwych przykładów.

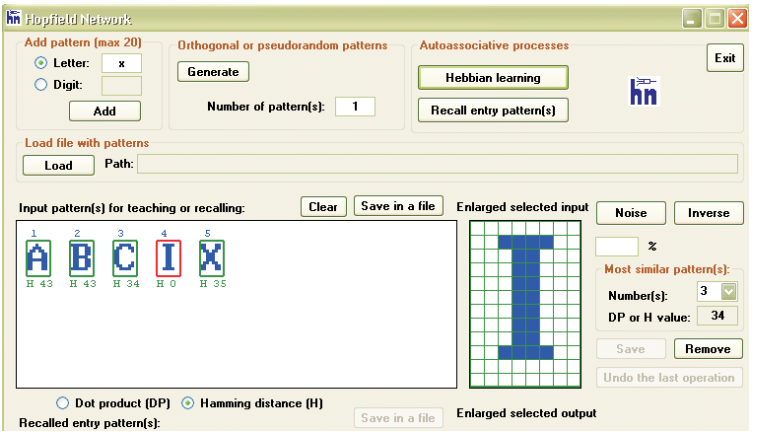



Rys. 11.15. Zestaw wzorców przygotowanych do zapamiętania w sieci Hopfielda

Za chwilę opiszę Ci, jak do programu można wprowadzać kolejne wzorce, na początku chciałbym jednak, żebyś się skupił na tym, co program robi i co z tego wynika – a na szczegóły techniczne przyjdzie czas za chwilkę.

Po wprowadzeniu (lub wygenerowaniu) wszystkich wzorców program **Example12b** tak ustawi parametry (współczynniki wagowe) wszystkich neuronów w sieci, żeby te właśnie obrazki stały się dla tej sieci punktami równowagi (atraktorami). O tym, na jakiej zasadzie to się odbywa oraz jak takie ustawianie wag przebiega – możesz przeczytać w mojej książce pt. *Sieci neuronowe*. Tutaj nie mogę Ci tego teraz opisywać, bo teoria uczenia sieci Hopfielda jest dość trudna i zmatematyzowana, a obiecywałem, że takich trudnych matematycznych rozważań w tej książce nie będzie. Nie musisz jednak znać szczegółów, wystarczy, że po zakończeniu wprowadzania do pamięci nowych wzorców klikniesz ukazany na rysunku 11.16 przycisk **Hebbian learning**. Spowoduje to automatyczne uruchomienie procesu uczenia, po którym sieć będzie już umiała przypominać sobie zapamiętane w niej wzorce. Jak wynika z napisu na przycisku – uczenie tej sieci odbywa się metodą Hebba, którą już wcześniej poznałeś, ale w tej chwili samym szczegółom procesu uczenia nie będziemy się przyglądali. Wystarczy, że zapamiętasz sobie, iż w trakcie tej nauki wytwarzane są takie wartości współczynników wag w całej sieci, by mogła ona osiągać stan równowagi w momencie, gdy na jej wyjściu pojawiają się obrazy odpowiadające zapamiętanym wzorcom.

Po nauce sieć jest gotowa do „egzaminu". Możesz go sam przeprowadzić. W tym celu najpierw wskazujesz (klikając myszką) wzorzec, którego stopień opanowania chcesz sprawdzić. Wzorce wraz z ich numerami są w tym momen-




Rys. 11.16. Wzorce do zapamiętania w sieci wprowadzasz w polu grupującym **Add pattern**

cie widoczne w oknie **Input pattern(s) for teaching or recalling,** możesz więc wybrać ten, który sieć ma sobie przypomnieć. Wybór (tj. zaznaczenie myszką) dowolnego z tych wzorców spowoduje, że jego powiększony obraz pojawi się w oknie zatytułowanym **Enlarged selected input**, a wybrana miara jego podobieństwa w stosunku do wszystkich innych wzorców pojawi się bezpośrednio pod miniaturowymi obrazkami wszystkich wzorców w oknie **Input pattern(s) for teaching or recalling** (patrz rys. 11.16). Stopień podobieństwa obrazków można mierzyć na dwa sposoby, dlatego poniżej okna **Input pattern(s) for teaching or recalling** są dwa pola do wyboru, opisane odpowiednio: **DP** – iloczyn skalarny albo **H** – odległość Hamminga. Co to znaczy dokładnie, opiszę Ci nieco dalej, teraz wystarczy, że zapamiętasz, iż **DP** mierzy, jak bardzo dwa obrazki są **podobne,** zatem duża wartość **DP** pod którymś wzorcem oznacza, że ten właśnie wzorzec będzie łatwo pomylić z tym, który Ty aktualnie wybrałeś. Z kolei odległość Hamminga jest (jak zresztą sugeruje jej nazwa) miarą **różnicy** między obrazami. Jeśli więc przy jakimś wzorcu pojawi się duża wartość miary **H**, to ten wzorzec jest bezpiecznie odległy od tego aktualnie wybranego przez Ciebie. Natomiast mała wartość **H** wskazuje na to, że ten wzorzec może się mylić z aktualnie przez Ciebie wybranym obrazkiem.

Gdy już wybrałeś obraz, którego zapamiętanie w sieci chciałbyś sprawdzić – możesz poddać „torturom" jego wzorzec, nietrudno jest bowiem przypomnieć sobie obrazek na podstawie jego idealnego wyobrażenia, co innego

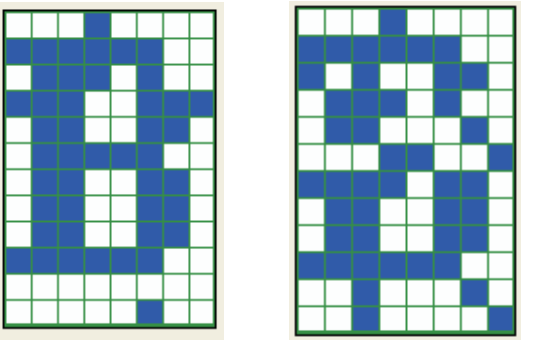
natomiast, gdy ten obraz zostanie losowo zniekształcony! O, wtedy właśnie sieć musi się wykazać swoimi umiejętnościami kojarzenia – i o to właśnie chodzi.

Wpisujesz więc w okienko po prawej stronie, oznaczone symbolem **%**, jaki procent punktów wzorca program ma zmienić, zanim zostanie on pokazany sieci, żeby go sobie spróbowała przypomnieć. W okienku oznaczonym symbolem **%** możesz podać dowolną liczbę od 0 do 99 i taki właśnie procent punktów wzorca zmieni program egzaminujący (w sposób przypadkowy) zanim rozpocznie egzamin. Żeby jednak wprowadzić do wzorca zaprogramowaną liczbę zakłóceń, musisz nacisnąć przycisk **Noise** (co oznacza szum albo zakłócenie). Żeby było jeszcze trudniej – możesz wprowadzić dodatkowo odwrócenie (negatyw) obrazka naciskając przycisk **Inverse.**

Wybrany i celowo zniekształcony wzorzec pojawi się w oknie **Enlarged selected input**. Przyjrzyj mu się: czy Ty sam byłbyś w stanie odgadnąć, z jakiego wzorca ten obrazek powstał? Dodatkowo możesz sprawdzić, czy Twój zmodyfikowany wzorzec nie stał się po tych wszystkich zmianach bardziej podobny do któregoś z obrazków „konkurencyjnych"? Przy ocenie sytuacji przydadzą Ci się miary jego „powinowactwa" z poszczególnymi wzorcami, które zostaną ukazane w postaci liczb położonych poniżej miniaturowych obrazków wszystkich wzorców w oknie **Input pattern(s) for teaching or recalling**.

**Radzę Ci**: nie stosuj na początku za dużych deformacji obrazu, bo trudno będzie rozpoznać w zmasakrowanym wzorcu jego pierwotny kształt – nie tylko sieci, ale i Tobie. Z doświadczenia mogę Ci zasugerować, że sieć nieźle sobie radzi przy zniekształceniach nie przekraczających 10%. Dobre efekty osiąga się też – na pozór paradoksalnie – przy bardzo dużej liczbie zmienionych punktów. Wynika to z faktu, że przy dużych liczbach zmienianych punktów obraz zachowuje swój kształt, dochodzi jednak do wymiany punktów białych na czarne i odwrotnie. Na przykład, po wybraniu liczby 99 jako procentu zmienianych punktów obraz zamienia się na swój idealnie dokładny negatyw. Tymczasem negatyw to w istocie ta sama informacja, co można łatwo w modelowanej sieci prześledzić. Przy liczbie zmienianych punktów nieco mniejszej od 99% powstaje też obraz dobrze dla sieci rozpoznawalny – negatyw z niewielkimi zmianami i sieć także „przypomina sobie" znajomy obraz bez kłopotu. Natomiast fatalnie złe wyniki daje próba odtwarzania wzorca przy zniekształceniach wynoszących od 30% do 70% – sieć coś tam niby sobie przypomina, ale zwykle wzorzec odtworzony zostaje bardzo późno (potrzeba wielu iteracji, zanim obraz się ustali), a rekonstrukcja wzorca odbywa się w sposób niedoskonały (pozostają spore zniekształcenia).

Tak więc na początku egzaminu tworzysz obraz mniej (na przykład popatrz na rys. 11.17, po lewej stronie) albo silniej zniekształconego wzorca

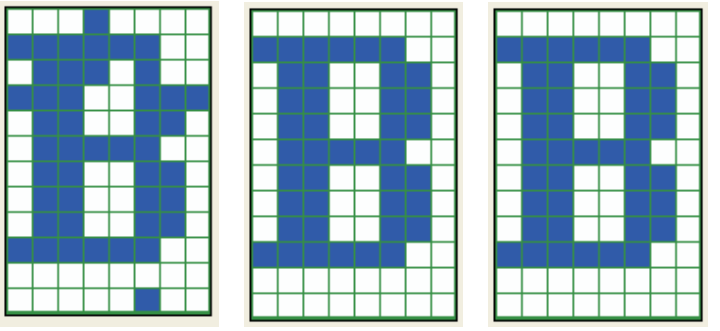


Rys. 11.17. Wzorce, od których rozpocznie się proces przypominania sobie wiadomości zapisanych w sieci: mało zniekształcony wzorzec litery B (po lewej stronie) oraz silnie zniekształcony wzorzec litery B (po prawej)

(rys. 11.17, po prawej stronie), który staje się punktem wyjścia do procesu przypominania sobie wzorca przez sieć.

Proces przypominania sobie wyuczonego wzorca polega na tym, że sygnały wyjściowe sieci podawane są na jej wejścia, tam (poprzez neurony) przekształcane są na nowe sygnały wyjściowe, które z kolei przez sprzężenia zwrotne kierowane są na wejścia sieci itd. Proces ten jest automatycznie przerywany, gdy w którejś kolejnej iteracji nie następują już żadne zmiany w sygnałach wyjściowych (sieć „przypomniała sobie" odpowiedni obraz). Zwykle tym przypomnianym obrazem jest obraz idealnego wzorca, którego zniekształconą wersję podałeś do sieci – ale niestety nie zawsze tak bywa.

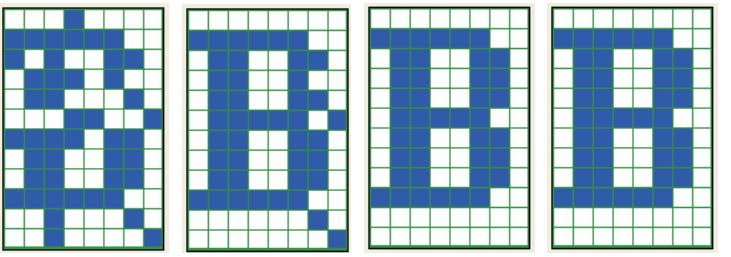
Jeśli obraz, od którego rozpoczyna się proces „przypominania", mało różni się od wzorca – przypomnienie może nastąpić praktycznie natychmiast. Ilustruje to rysunek 11.18, na którym pokazałem, jak sieć przypomniała sobie poprawny kształt litery **B**, wychodząc od mało zniekształconego wzorca pokazanego na rysunku 11.17 po lewej stronie. Kolejne obrazki na rysunku 11.18 (i na wszystkich dalszych w tym rozdziale) pokazują (w kolejności od lewej



Rys. 11.18. Szybkie odtwarzanie mało zniekształconego wzorca w sieci Hopfielda pracującej jako pamięć skojarzeniowa

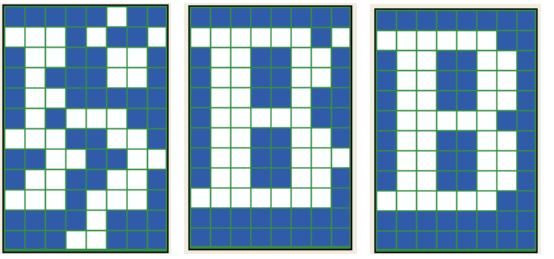
do prawej) obliczane i wyprowadzane z modelu zestawy sygnałów wyjściowych z badanej sieci. Na rysunku 11.18 widać, że sygnał wyjściowy z sieci osiągnął pożądany stan (idealne odtworzenie wzorca) już po jednej iteracji.

Nieco bardziej złożone procesy przebiegały w badanej sieci podczas odtwarzania silnie zniekształconego wzorca, pokazanego na rysunku 11.17 po prawej stronie. Tutaj sieć potrzebowała dwóch iteracji, żeby osiągnąć końcowy sukces (rys. 11.19).



Rys. 11.19. Odtwarzanie silnie zniekształconego wzorca w sieci Hopfielda

Jedną z ciekawych cech sieci Hopfielda, którą poznasz podczas samodzielnych eksperymentów z programem, jest jej zdolność do zapamiętywania zarówno oryginalnych sygnałów związanych z kolejnymi wzorcami, jak i sygnałów stanowiących **negatywy** zapamiętanych wzorców. Można wykazać matematycznie, że tak się dzieje **zawsze**. Każdy etap procesu uczenia sieci, który prowadzi do zapamiętania pewnego wzorca, automatycznie powoduje także powstanie w sieci atraktora odpowiadającego negatywowi wzorca. W związku z tym proces odtwarzania wzorców może się zakończyć zarówno odnalezieniem oryginalnego sygnału – jak i odnalezieniem jego negatywu. Ponieważ negatyw zawiera dokładnie tę samą informację, co oryginalny sygnał (jedyna różnica polega na tym, że w miejscach, gdzie oryginalny sygnał ma wartość +1 w negatywie występuje –1 i odwrotnie), w związku z tym



Rys. 11.20. Odtworzenie przez pamięć skojarzeniową
negatywu zapamiętanego wzorca

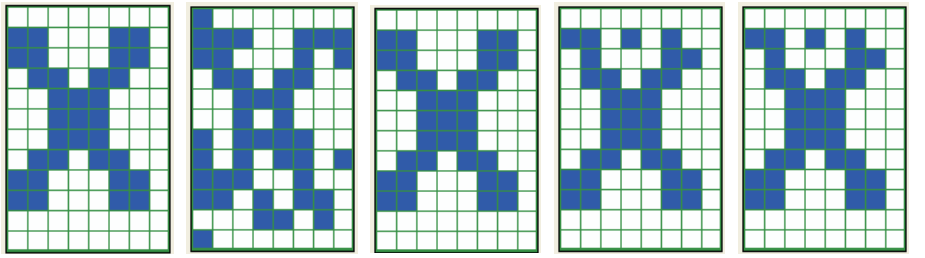
uważa się odnalezienie przez sieć negatywu zakłóconego wzorca – także za sukces. Przykładowo na rysunku 11.20 możesz obejrzeć proces odtwarzania bardzo zniekształconego wzorca litery **B**, który zakończył się znalezieniem negatywu tej litery.

## 11.6. Jak działa program pozwalający Ci samodzielnie badać działanie sieci Hopfielda?

Opisane wyżej zjawiska (i wiele, wiele innych) możesz sam zobaczyć i przebadać posługując się programem **Example12b.** Program ten rozpoczyna pracę od tego, że przyjmuje od Ciebie (lub – na Twoje życzenie – sam generuje) wzorce, które sieć ma zapamiętać. Są to – jak już wiesz – obrazy liter lub cyfr podawanych z klawiatury – albo obrazy abstrakcyjne, tworzone samodzielnie przez program, o czym jeszcze za chwilę porozmawiamy. W celu lepszej widoczności miniatur wzorców na ekranie, a także ze względu na fakt, że pamięć skojarzeniowa (jak każda pamięć) ma ograniczoną pojemność, program **Example12b** ogranicza maksymalną liczbę zapamiętywanych w sieci wzorców wejściowych do 20. Ty możesz podać tyle wzorców, ile tylko chcesz, jednak w praktyce dobrze zrobisz ograniczając swoje „apetyty". Im mniej wzorców, tym szybciej sieć się uczy i tym szybciej działa (możesz wykonać więcej doświadczeń) oraz – co jeszcze ważniejsze – mniej się myli. Powyżej 16 wzorców sieć zaczyna mieszać i nakładać na siebie zgromadzone wiadomości (ze względu na ograniczoną liczbę neuronów zapamiętujących podawane wzorce), w wyniku czego przy odtwarzaniu wiedzy z sieci będą widoczne znaczne błędy. Błędy te manifestują się w ten sposób, że w odtwarzanych wzorcach pojawią się tzw. „przesłuchy", polegające na tym, że w odtwarzanych przez sieć obrazach zapamiętanych wzorców pojawiają się elementy innych wzorców. Efekt pojawienia się takich „przesłuchów" zobaczyć możesz na rysunku 11.21. Na rysunku tym (i na dalszych w tym rozdziale) zastosowano trochę inny, niż wcześniej stosowany, sposób przedstawiania przebiegu działania sieci. Kolejno od lewej do prawej pokazane są:

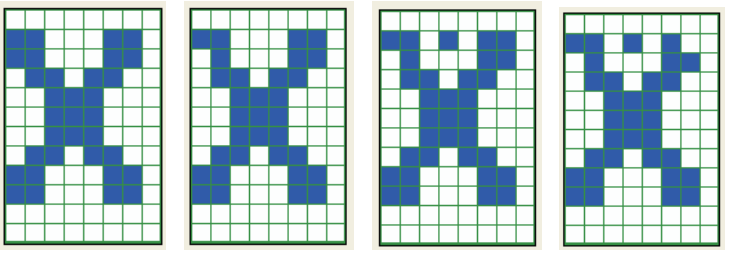
- odtwarzany wzorzec bez zniekształceń,
- wzorzec po wprowadzeniu do niego celowych zniekształceń przez program Example12b oraz
- kolejne etapy odtwarzania wzorca przez sieć.

Taki sposób prezentacji będzie użyteczny, gdy zaczniemy operować licznymi wzorcami o różnych kształtach – możliwość porównania oryginału wzorca (skrajnie po lewej) i ostatecznego wyniku pracy sieci „przypominającej sobie" wzorzec (skrajnie po prawej) zazwyczaj starczy za komentarz.



Rys. 11.21. Zniekształcenie wzorca na skutek przesłuchów

Analizując rysunek 11.21, można dobitnie stwierdzić, że przeładowanie sieci nadmierną liczbą zapamiętanych wzorców powoduje zawsze nieusuwalne zakłócenie jej pracy. Zakłócenie to polega na tym, że przy odtwarzaniu wzorca pojawia się on wtedy w postaci zniekształconej, wyraźnie odmiennej od oryginału, który był użyty w trakcie procesu uczenia. Zjawisko to daje się zaobserwować nawet wtedy, gdy proces odtwarzania zaczyna się od sygnału praktycznie nie zniekształconego (patrz rys. 11.22), ponieważ w strukturę zapamiętanego w sieci znaku na trwale wpisały się ślady pamięciowe pochodzące od innych wzorców.

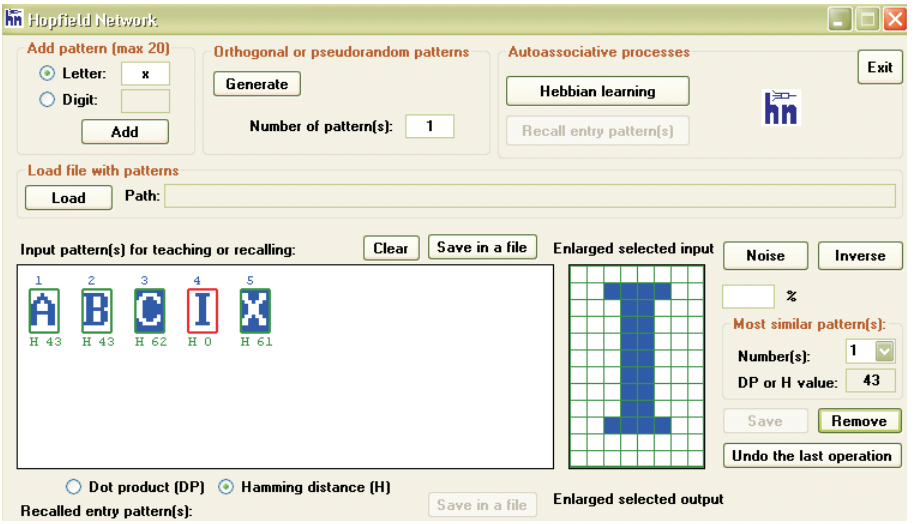


Rys. 11.22. Wzorzec zniekształcony na skutek przesłuchów nie daje się odtworzyć poprawnie, nawet jeśli na wejściu zostanie podany w postaci nie zakłóconej

Warto zauważyć jedno ciekawe zjawisko, które ujawniło się na rysunku 11.21. Otóż w pierwszej iteracji, zaraz po podaniu do sieci silnie zniekształconego wzorca litery X, program odtworzył **idealny** obraz wzorca, a dopiero potem dalsze przetwarzanie spowodowało zakłócenie odtworzonego wzorca „echem" wspomnień o innych zapamiętanych przez sieć wzorcach. Odpowiada to znanej Ci zapewne z własnego doświadczenia sytuacji, że rozwiązując trudny problem przekonujesz się (często poniewczasie), że pierwszy pomysł jaki Ci przyszedł do głowy, był w istocie najlepszy, natomiast potem, rozważając i roztrząsając problem na wiele różnych sposobów, doprowadziłeś

(sam siebie) do tego, że uzyskałeś rozwiązanie o wiele gorsze niż to, które Ci intuicja podpowiadała na początku.

Powróćmy jednak do dokładniejszego opisu sposobu działania programu **Example12b.** Jak już wspomniałem, po uruchomieniu program zbiera dane, które zostaną zapamiętane w sieci jako wzorce. Prosi w związku z tym o podanie wzorca (rys. 11.23), który **może** (ale nie musi) zostać zapamiętany w sieci. Wzorce do zapamiętania w sieci wprowadzisz w polu grupującym **Add pattern (max 20)**. Możesz dodać literę (wpisując ją do pola edycyjnego **Letter:**) lub cyfrę (wpisując ją do pola edycyjnego **Digit:**). Wybór wpisanej przez Ciebie litery lub cyfry potwierdź klikając przycisk **Add**, który spowoduje, że miniatura dodanej przez Ciebie litery lub cyfry pojawi się w oknie graficznym **Input pattern(s) for teaching or recalling:.** Pole grupujące **Add pattern (max 20)** pozwoli Ci na dodanie do sieci maksymalnie 20 wzorców (rys. 11.23). Program dba o to, żebyś miał dokładną kontrolę nad wprowadzanymi do pamięci wzorcami, dlatego po naciśnięciu klawisza odpowiedniej litery lub cyfry pokazuje Ci dokładną strukturę jej obrazka w obrębie pola **Enlarged selected input.**




Rys. 11.23. Sposób wprowadzania danych do pamięci autoasocjacyjnej z możliwością modyfikacji lub odrzucenia każdego z wprowadzonych ręcznie znaków

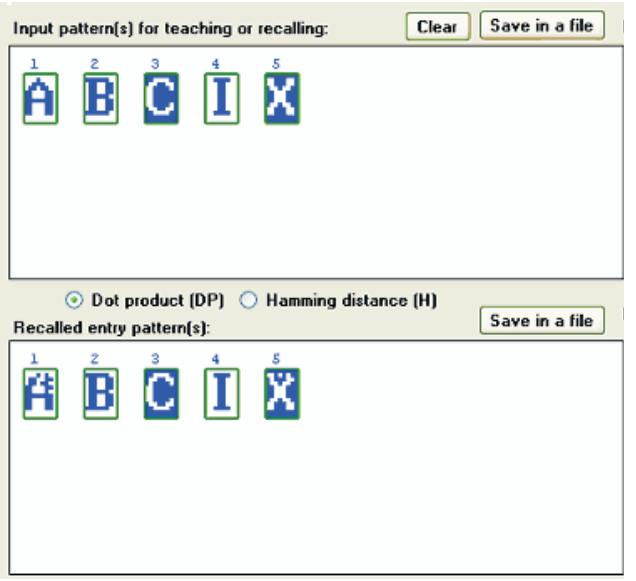
Po wprowadzeniu wzorców do zapamiętania (wszystkie one pojawiają się wtedy w oknie **Input pattern(s) for teaching or recalling**) możesz każdy z wzorców zaznaczyć myszką i dokonać jego inwersji (klikając przycisk **Inverse** oraz **Save**) lub usunąć z sekwencji wzorców uczących (przycisk **Remo-**

**ve**). To ostatnie bywa jedynym wyjściem, jeśli po podaniu kolejnego nowego wzorca wykryjesz, że jest on zbyt podobny do wzorców już wcześniej wprowadzonych.

Możliwe są także automatycznie generowane wzorce (będzie o nich dalej mowa), mają one jednak niestety mało czytelne kształty, więc może na początku nie stosuj ich zbyt wiele. Najlepiej zacznij od kilku liter (ja stosowałem zestaw **A, B, C, I, X**). Po wpisaniu każdej z liter w polu edycyjnym **Letter:** naciśnij przycisk **Add**.

Po zakończeniu wprowadzania (i ewentualnej modyfikacji) sekwencji wzorców do zapamiętania w sieci (widocznych w oknie graficznym **Input pattern(s) for teaching or recalling**) kliknij przycisk **Hebbian learning.** Przycisku **Hebbian learning** możesz używać często, zwłaszcza kiedy chcesz uczyć sieć jakiejś innej sekwencji wzorców. Po kliknięciu przycisku **Hebbian learning** niemal jednocześnie uaktywniony zostaje przycisk **Recall entry pattern(s),** oznaczający zakończenie procesu uczenia i gotowość sieci do poddania się egzaminowi.

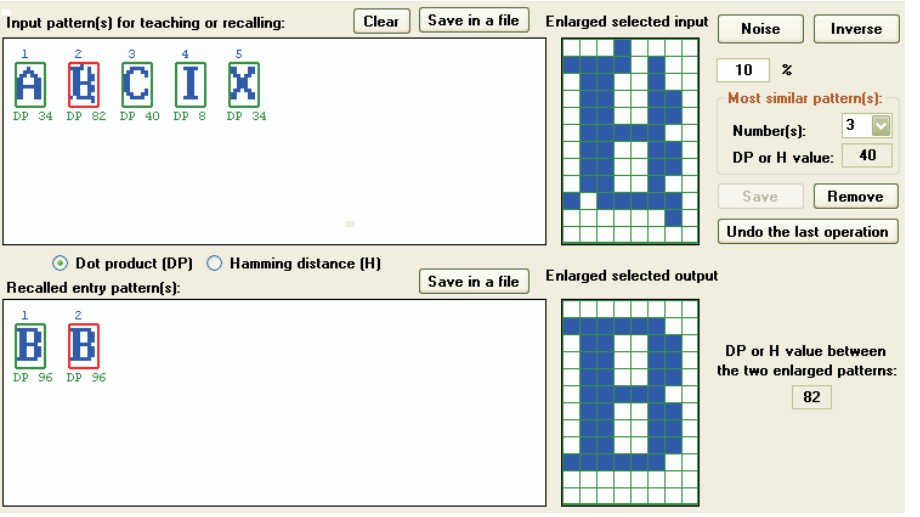
Możesz teraz rozpocząć sprawdzanie, jak sieć potrafi odtworzyć każdy z zapamiętanych w niej wzorców. Kliknięcie przycisku **Recall entry pattern(s)** – gdy żaden z wzorców w oknie **Input pattern(s) for teaching or recalling** nie jest zaznaczony – powoduje uruchomienie procedury odtwarzającej każdy z wzorców wejściowych, których miniatury widnieją w oknie **Input pattern(s) for teaching or recalling.** Wzorce odtworzone pojawią się w oknie **Recalled entry pattern(s)**.




Rys. 11.24. Odtworzenie wszystkich wprowadzonych do sieci wzorców

Obserwacja, jak sieć sobie radzi z zapamiętywaniem tych liter i ich odtwarzaniem, jest bardzo ciekawa. Okazuje się na przykład, że sieć dobrze i skutecznie przypomina sobie **B**, nieco gorzej **C, I** oraz **X**, natomiast regularnie pojawiają się „przesłuchy" przy odtwarzaniu **A** (rys. 11.24).

Chyba zauważyłeś, że po kliknięciu na dowolnym wzorcu znajdującym się w oknie **Input pattern(s) for teaching or recalling** uaktywnione zostają przyciski **Noise** i **Inverse**, po prawej stronie okna **Enlarged selected input.** Pole edycyjne z symbolem %, znajdujące się pod przyciskiem **Noise**, służy do wpisywania procentu punktów, jaka we wzorcu ma zostać zmieniona. Jak już było wspomniane, możesz w nim wpisać dowolną liczbę od 0 do 99. Po kliknięciu przycisku **Noise** program zmieni w sposób losowy tyle punktów wzorca, ile sobie zażyczyłeś oraz wstawi powiększoną miniaturę zniekształconego wzorca w oknie **Enlarged selected input** (rys. 11.25)**.** Jeżeli klikniesz przycisk **Save** (który jest położony poniżej okna **Enlarged selected input),** to ten zaszumiony wzorzec zostanie wstawiony do sekwencji wzorców do odtwarzania, która to sekwencja widoczna jest w oknie **Input pattern(s) for teaching or recalling.**

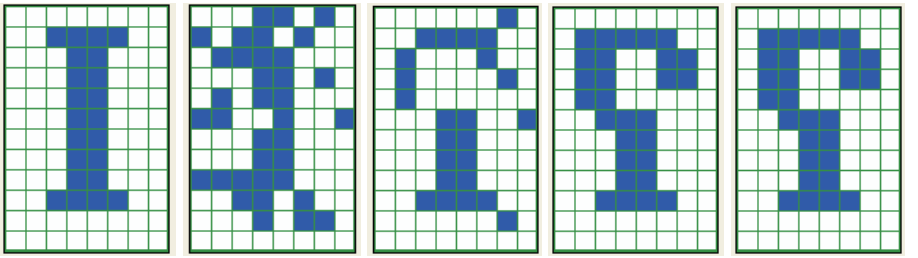



Rys. 11.25. Możliwość porównania obrazów w oknach: **Enlarged selected input** i **Enlarged selected output** – oraz miary podobieństwa między nimi (**DP or H value between the two enlarged patterns**)

Obok przycisku **Noise** znajduje się przycisk **Inverse**, o nieco podobnym działaniu. Przycisk **Inverse** wykonuje negatyw zaznaczonego przez Ciebie wzorca, tj. zamienia niebieskie piksele na białe, a białe na niebieskie. Zmia-

ny dokonane na wzorcu po kliknięciu przycisku **Inverse** – tak jak to było w przypadku przycisku **Noise** – również trzeba zatwierdzić kliknięciem przycisku **Save**. Rezultat kliknięcia przycisku **Save** jest identyczny w obu wspomnianych przypadkach. Jeżeli chcesz cofnąć ostatnio wprowadzoną zmianę, możesz użyć przycisku **Undo the last operation,** który cofa rezultat operacji **Save.** Przycisk **Undo the last operation** cofa również rezultat operacji usuwania wybranego wzorca (przycisk **Remove)**. Ogólnie, przycisk **Undo the last operation** anuluje ostatnio wykonaną operację: **Save** lub **Remove**.

Zagadnienie podobieństwa lub braku podobieństwa zapamiętywanych wzorców jest zagadnieniem o kluczowym znaczeniu, dlatego poświęcimy mu odrobinę więcej uwagi. Otóż zauważyłeś (przy okazji dyskusji rysunków 11.21 i 11.22), że niekiedy wzorce sygnałów nakładają się na siebie, co powoduje „przesłuchy". Takie zjawiska zachodzą tym silniej, im bardziej podobne do siebie są zapamiętywane wzorce. Jest to logiczne – przy podobnych wzorcach ślady pamięciowe (znajdujące się – wszystkie! – w tej samej sieci) bardziej nakładają się na siebie. Jeśli ten sam neuron raz ma wyświetlać +1, bo jest częścią zapamiętywanej litery **A,** czasem –1, bo jest częścią litery **B**, a potem znowu +1 w ramach przypominania sobie przez sieć litery **C** – to sprawa staje się kłopotliwa i trudna. W przypadku, gdy zapamiętywane obrazy silnie różnią się od siebie – sprawa jest prostsza, bo mniej jest tych „konfliktowych" punktów. Jeśli jednak są bardzo podobne, to ich zapisy w postaci składników wag poszczególnych neuronów tak bardzo mieszają się ze sobą w pamięci neuronowej, że w gotowej sieci zaczyna być problematyczne prawidłowe odtworzenie któregokolwiek z zapamiętanych obrazów (rys. 11.26).



Rys. 11.26. Efekt mieszania się śladów pamięciowych przy silnie mieszających się sygnałach

Dlatego chcąc obserwować działanie sieci we w miarę korzystnych warunkach musisz postarać się dobrać – przynajmniej na początku – **bardzo dobrze rozróżnialne wzorce**. Program będzie Cię w tym wspomagał na dwa sposoby.

Pierwszy sposób polega na tym, że podczas wprowadzania nowych wzorców program natychmiast wylicza i podaje stopień podobieństwa nowego

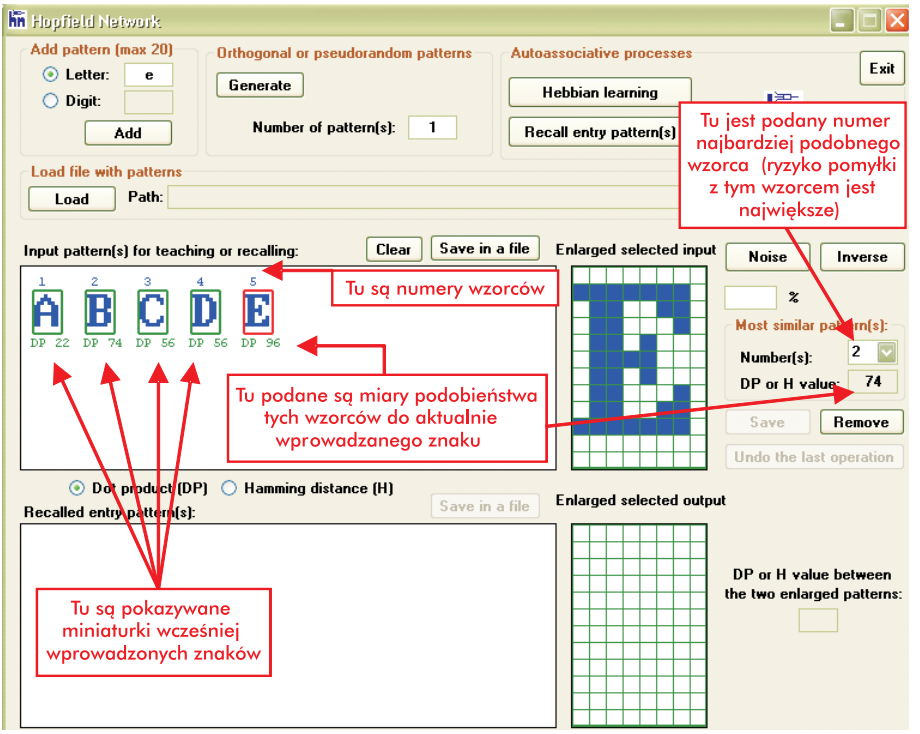
obrazu do obrazów już wcześniej zapamiętanych. Możesz więc na bieżąco kontrolować efekty nakładania się wzorców podczas „ręcznego" wprowadzania znaków z klawiatury. Jest to łatwe i proste, ponieważ kliknięcie myszką na dowolnej miniaturze wprowadzonego wcześniej wzorca (a widocznej w oknie graficznym **Input pattern(s) for teaching or recalling**) powoduje, że pod każdym zminiaturyzowanym obrazkiem (zaopatrzonym w swój numer kolejny) pokazują się liczby określające **podobieństwo aktualnie zaznaczonego obrazka z każdym spośród już wcześniej wprowadzonych wzorców.** Pamiętasz, że ta miara może być pokazana jako **DP** (wtedy im większa wartość, tym gorzej – bo to oznacza duży stopień podobieństwa nowego wzorca do określonego wcześniej wprowadzonego wzorca) albo jako **H** (wtedy im większa wartość, tym lepiej – bo to oznacza duży odstęp nowego wzorca w stosunku do określonego wcześniej wprowadzonego wzorca).

Dodatkowo, w oknie **Enlarged selected input** – zostanie przedstawiony duży widok dowolnego zaznaczonego przez Ciebie wzorca, a po prawej stronie tego okna, tuż pod przyciskami **Noise** i **Inverse** – znajduje się pole grupujące **Most similar pattern(s),** podające Ci informacje, do którego z już istniejących wzorców ten obrazek jest najbardziej podobny. Korzystając z tego pola możesz się dowiedzieć, gdzie leżą ewentualne zagrożenia, bo program wyświetli Ci informację na ten temat (w postaci numeru odpowiedniego wzorca, z którym grozi pomylenie aktualnie wprowadzanego obrazka). Numer takiego „potencjalnie kolidującego" wzorca pokaże się w polu **Numbers, a** w polu edycyjnym **DP or H value:** zostanie wyświetlona wartość aktualnie wybranej miary podobieństwa (tj. albo maksymalna wartość iloczynu skalarnego, albo minimalna odległość Hamminga).

## 11.7. Kilka ciekawych przykładów

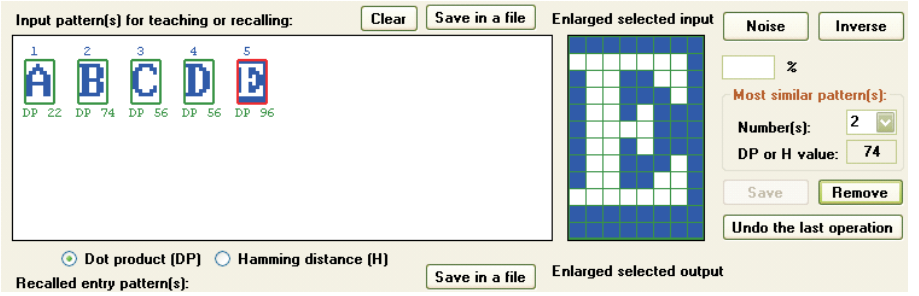
Zdaję sobie sprawę, że podany wyżej opis obsługi i działania programu **Example12b** mógł Cię nieco znużyć – dlatego teraz przyszedł czas na nagrodę. Zaczniemy się wspólnie bawić tym programem, żebyś zobaczył, jakie fajne rzeczy potrafi robić, a potem już sam będziesz mógł prowadzić dalsze badania, opierając się na praktycznych wskazówkach, jakie Ci teraz podam.

Przeanalizujmy wspólnie sytuację pokazaną na rysunku 11.27. Jak widać w górnej części ekranu – wcześniej wprowadziłem już wzorce **A**, **B**, **C** i **D**. Aktualnie wprowadzany wzorzec litery **E** okazuje się stosunkowo słabo zróżnicowany w stosunku do pozostałych. Miara jego podobieństwa do wzorca liter **A** wynosi 22 (to jeszcze całkiem przyzwoity wynik), ale ze wzorcem litery **B** wiąże go siła 74 jednostek, a z pozostałymi wzorcami **C** i **D** siła 56.




Rys. 11.27. Sposób informowania o stopniu podobieństwa nowego obrazu do już zapamiętanych wzorców

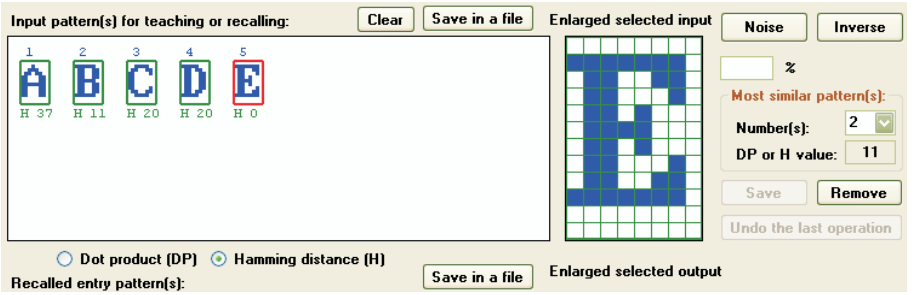
Po prawej stronie okna zobaczysz (w polu **Most similar pattern(s)**), jak program ostrzega, że wprowadzany wzorzec będzie się mylił ze wzorcem litery **B** (nic dziwnego...). Oczywiście, decyzja należy do Ciebie, ja jednak w moich badaniach taki wzorzec bym zdecydowanie odrzucił. Warto może zauważyć, że wzięcie inwersji (czyli negatywu rozważanego feralnego




Rys. 11.28. Wzorzec w inwersji wykazuje dokładnie ten sam stopień podobieństwa do wcześniej podanych wzorców, co wzorzec oryginalny

wzorca) nie zmienia stopnia jego podobieństwa do rozważanych wcześniej wzorców (rys. 11.28).

Wróćmy do szczegółów technicznych. Ta miara stopnia podobieństwa nowego obrazka do poszczególnych już wcześniej zdefiniowanych wzorców, która proponowana jest przez program jako podstawowy miernik – obliczana jest w programie jako tak zwany **iloczyn skalarny** (**Dot product, DP**) wektorów opisujących te sygnały w przestrzeni sygnałów wejściowych. Dokładniej – brana jest tu pod uwagę bezwzględna wartość iloczynu skalarnego, ale naprawdę nie musisz się zastanawiać nad tym, co to znaczy. Przypomina o tym kropka zaznaczona w polu wyboru, obok napisu **Dot product (DP)** widocznego w środku ekranu. Wystarczy jednak, że naciśniesz drugi (znajdujący się obok) przycisk wyboru, oznaczony napisem **Hamming distance** (**H**) – a program przestawi się na inny sposób oceny podobieństw wzorców i będzie wyznaczał tak zwane **odległości Hamminga** (rys. 11.29).




Rys. 11.29. Miary podobieństwa sygnałów wyrażone przez odległości Hamminga

Tak więc przyciski wyboru: **Dot product (DP)** oraz **Hamming distance (H)** decydują o tym, która wielkość – DP czy H – zostanie aktualnie wyświetlona pod każdą miniaturą wzorca.

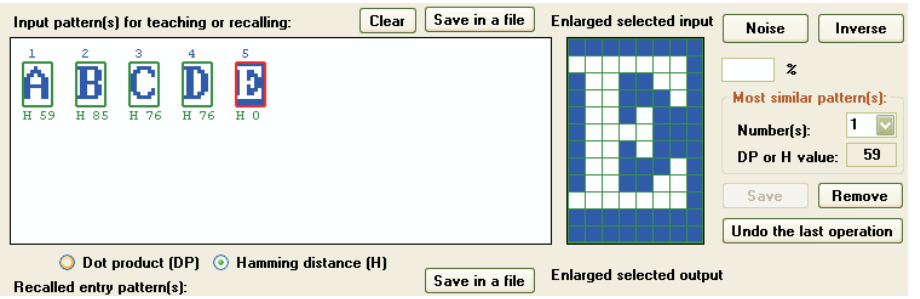
Zauważ, że na rysunku 11.29 liczby pod poszczególnymi wzorcami są odmienne – teraz ten wzorzec, który jest najbardziej podobny, wykazuje **najmniejszą** odległość od tego obrazka, który zaznaczyłeś. Wiesz jednak, jak to interpretować, ponieważ czarna kropka widnieje teraz w polu wyboru obok napisu **Hamming distance** (**H**), a nie – jak poprzednio w polu wyboru obok napisu **Dot product** (**DP**)[2].

Po co są takie dwie możliwości w jednym programie?

[2] Iloczyn skalarny pokazuje, w jakim stopniu nowy obrazek powtarza elementy już wcześniej występujące w zaakceptowanych wzorcach, natomiast odległość Hamminga pokazuje, w ilu punktach dwa obrazki różnią się od siebie.

Otóż w zależności od sytuacji te dwie miary dostarczają **innych** informacji i mogą być użyteczne do dwóch różnych celów.

Generując nowe obrazki, lepiej opierać się na obserwacji (i maksymalizacji) iloczynu skalarnego. Odległość Hamminga może być myląca, ponieważ na przykład jest wrażliwa na operację brania inwersji wzorca (patrz rys. 11.30 i porównaj go z rys. 11.27., 11.28 i 11.29), co przy wprowadzaniu i ocenianiu nowych wzorców nie powinno mieć miejsca.

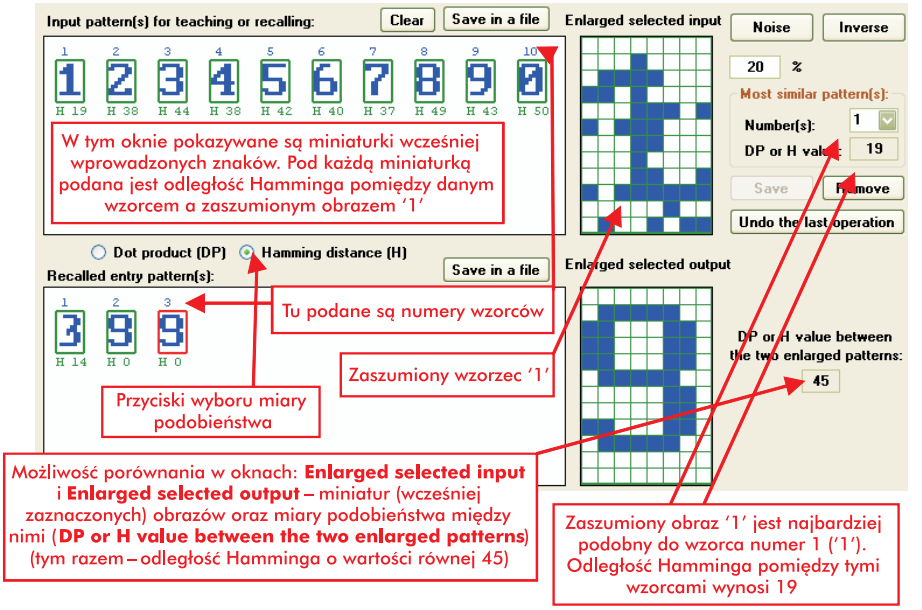



Rys. 11.30. Zmiana wartości odległości Hamminga przy inwersji znaku

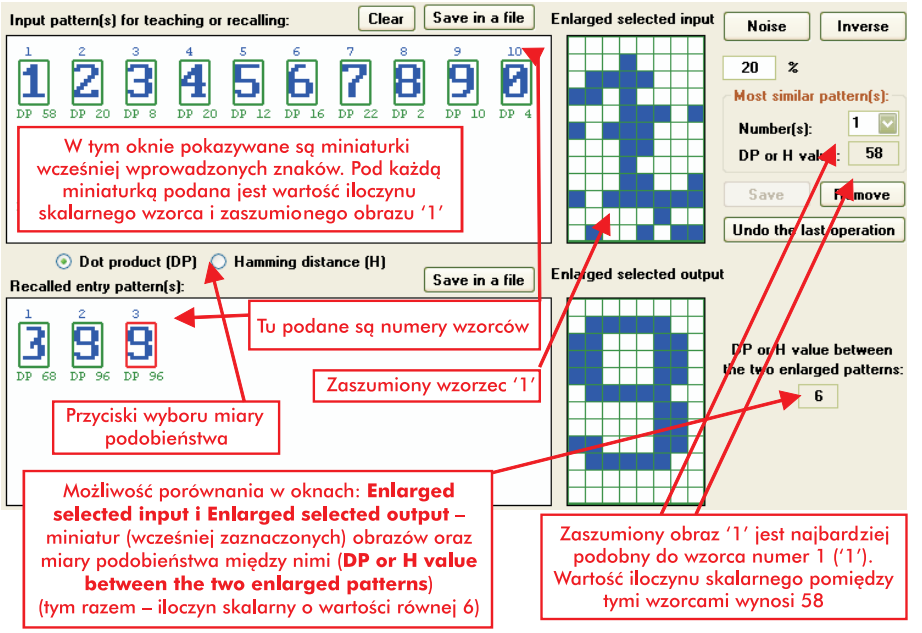
Natomiast podczas obserwacji pracy sieci, kiedy będziesz śledził jej wysiłki zmierzające do przypomnienia sobie jakiegoś wzorca – bardziej przydatna będzie odległość Hamminga. Pozwoli Ci ona oceniać, czy proces poszukiwań jest na dobrej drodze, czy sieć zbliża się, czy oddala od prawidłowego wzorca. To bywa bardzo emocjonujące – zwłaszcza gdy uda się zaobserwować bardzo ciekawe zjawisko – ucieczki sieci od właściwego wzorca i „przypominanie sobie" zupełnie innej zapamiętanej informacji. Obejrzyj przykład podany na rysunku 11.31.

Przykład ten odpowiada sytuacji wprowadzenia do sieci bodźca zakłóconego. W oryginalnym sygnale nr 1 dokonano 20 losowych zmian, w związku z tym stał się on mniej podobny do swojego własnego wzorca i także stał się bardzo mało podobny do wszystkich innych wzorców. Wyraża to zarówno wygląd samego sygnału, jak i potwierdzają podane przez program wartości odległości Hamminga.

Od tego momentu startuje proces dynamiczny związany z działaniem sprzężeń zwrotnych występujących w sieci. Najpierw sieć wytwarza obraz, który niebezpiecznie zbliża się do obrazu **3**, ale potem widać wyraźnie, że następuje „wessanie" sieci przez atraktor związany z wzorcem **9.** Zwróć uwagę na liczby pojawiające się poniżej zestawu wzorców zarówno w oknie graficznym **Input pattern(s) for teaching or recalling**, jak i w oknie **Recalled entry pattern(s)**. Podają one odległości Hamminga (a na rysunku 11.32 – wartości



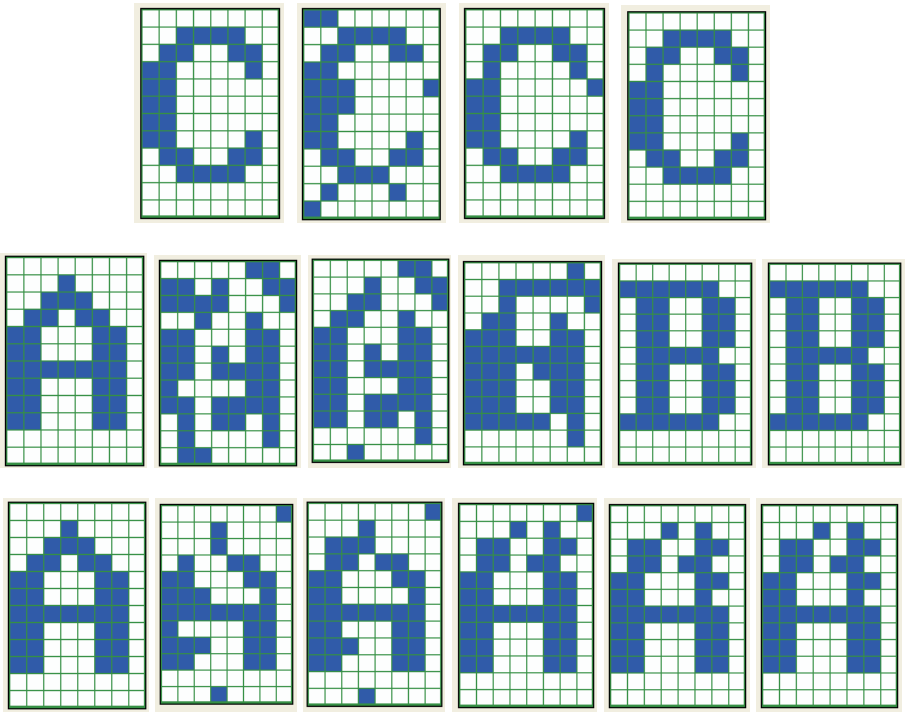

Rys. 11.31. Przebieg poszukiwania wzorca zakończony niepowodzeniem, obserwowany przy zaznaczonej mierze Hamminga




Rys. 11.32. Przebieg poszukiwania wzorca przy zaznaczonej mierze iloczynu skalarnego, zakończony niepowodzeniem

iloczynów skalarnych) między sygnałem aktualnie występującym na wyjściu sieci a wszystkimi rozważanymi wzorcami. Rysunek pomoże Ci w orientacji, które sygnały odpowiadają którym zestawom odległości. Porównaj wartości odległości Hamminga pomiędzy zaszumionym wzorcem ‘1’ a każdym wzorcem zarówno w oknie **Input pattern(s) for teaching or recalling**, jak i w oknie **Recalled entry pattern(s)**. Zauważ na początku, że odległość Hamminga między poprawną wersją wzorca nr 1 a  a pokazywanym na wejściu sieci sygnałem wynosi 19. Do innych wzorców jest bezpiecznie daleko – od 37 do 50 jednostek. Więcej jednostek odległości melduje program tam, gdzie sygnał i bodziec są mało podobne (**0** i **1)**, mniej tam, gdzie jednak pewnego podobieństwa można się doszukać, więc wszystko jest sensowne i logiczne.

Doświadczenie, które Ci wyżej opisałem, możesz z łatwością powtórzyć na swoim komputerze. Możesz także uzyskać setki podobnych wyników z innymi wzorcami i z innymi sygnałami. Warto się nad nim trochę potrudzić, ponieważ jego przeprowadzenie, a zwłaszcza przemyślenie wyników daje podstawy do bardzo ciekawych wniosków. Może właśnie na tej zasadzie rodzą się i w Twoim mózgu nowe pomysły i nieoczekiwane skojarzenia?



Rys. 11.33. Powodzenie przy odtwarzaniu wzorca litery **C** i dwie formy niepowodzenia przy odtwarzaniu przez sieć neuronową wzorca litery **A**

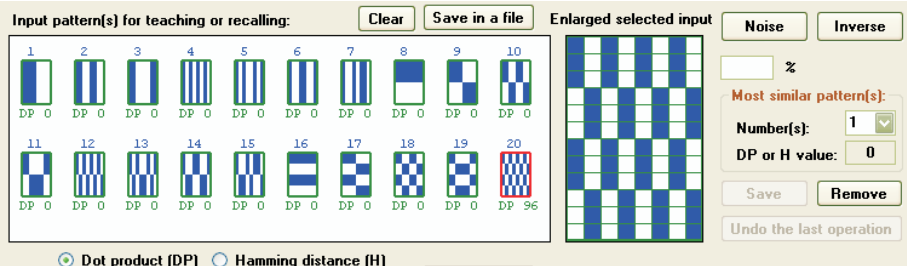
Doświadczenia wykonywane z siecią Hopfielda pokażą Ci, że zadanie zadaniu nierówne. Niektóre wzorce dają się odtworzyć łatwo i bez większych kłopotów, a inne uporczywie się psują (patrz rys. 11.33). Dlatego w kolejnym podrozdziale opowiem Ci o takich wzorcach, które się będą dobrze odtwarzały (chociaż być może pożytek z tego będzie niezbyt wielki).

## 11.8. Jak i po co można korzystać z automatycznej generacji wzorców dla sieci Hopfielda?

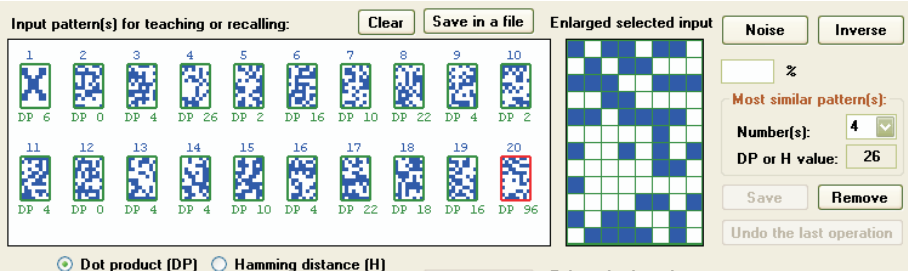
Jak już wyżej Ci napisałem, podczas konstruowania wzorców, które sieć ma zapamiętać, powinieneś się posługiwać wartościami iloczynów skalarnych, żeby uzyskać zestaw wzorców mających możliwie mały stopień wzajemnego „przesłuchu". Wiesz o tym, że przyzwoicie skonstruowany zestaw wzorców musi charakteryzować się jak najmniejszymi wartościami tych iloczynów (w układzie „każdy z każdym"). Ideałem są więc wzorce mające **zerową** wartość iloczynu skalarnego. Taki zestaw wzorców z zerowymi wartościami iloczynów skalarnych we wszystkich parach typu „każdy z każdym" nazywany jest w matematyce **układem ortogonalnym.** Układ ortogonalny ma cały szereg znakomitych właściwości teoretycznych, wprawiających matematyków w stan graniczący z ekstazą. Można o tym napisać kilka grubych tomów wyłącznie formułami i wzorami, przetykanych takimi słowami, jak *ortonormalna baza układu współrzędnych, inwariantność przekształcenia, dekorelacja zmiennych, transformacja składowych kanonicznych* i *diagonalizacja macierzy kowariancji* (to wszystko naprawdę ma sens i ma głęboki związek z używanym tu programem), ale Ty teraz i tutaj **naprawdę nie musisz tego wiedzieć.** Natomiast musisz wiedzieć, że przy wzorcach zbliżających się (przynajmniej) do opisanego ideału proces uczenia sieci i proces jej eksploatacji – przebiegać będą w sposób maksymalnie korzystny.

Niestety, własnoręczne skonstruowanie zestawu ortogonalnych wzorców jest praktycznie niemożliwe. Możesz próbować zbliżyć się do tego ideału, zawsze jednak obrazy znaków generowanych na klawiaturze komputera okazują się w jakimś stopniu skorelowane, czyli ich iloczyn skalarny będzie różny od zera. Możesz jednak powierzyć to zadanie programowi **Example12b.** Program ten potrafi utworzyć ortogonalne wzorce, dzięki czemu bardzo efektywnie wspomaga on Twoją pracę. W górnej części ekranu znajduje się pole grupujące **Orthogonal and pseudorandom patterns**, a w nim – pole edycyjne **Number of pattern(s)**, w którym należy wpisać liczbę wzorców, które program ma dla Ciebie automatycznie wygenerować. Pamiętaj, aby potwierdzić swoją decyzję naciśnięciem przycisku **Generate**. Liczba podana przez Ciebie w polu edycyjnym **Number of pattern(s)** musi być tak dobrana, aby

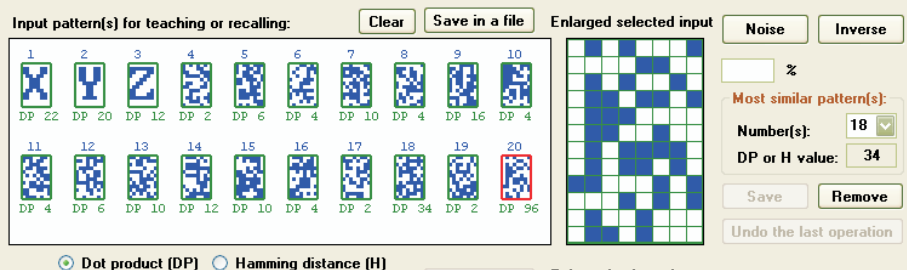
łącznie z innymi dodanymi wcześniej literami i cyframi – nie była przekroczona wartość maksymalna programu (tj. 20 wzorców). Naciśnięcie przycisku **Generate** (podobnie, jak to było z przyciskiem **Add**) spowoduje, że w oknie graficznym **Input pattern(s) for teaching or recalling:** pojawią się miniatury (tylu – ile sobie zażyczyłeś) wygenerowanych przez program wzajemnie ortogonalnych wzorców. Automatycznie tworzyć i dodawać wzorce do sieci możesz zacząć zaraz na początku (rys. 11.34) albo możesz spróbować układu kombinowanego – najpierw Ty podasz jeden albo nawet kilka wzorców, a potem dopiero oddasz sterowanie maszynie (rys. 11.35, 11.36).




Rys. 11.34. Zestaw ortogonalnych wzorców wygenerowany od początku wyłącznie automatycznie




Rys. 11.35. Automatycznie generowany zestaw wzorców pseudolosowych (zapoczątkowany znakiem X)

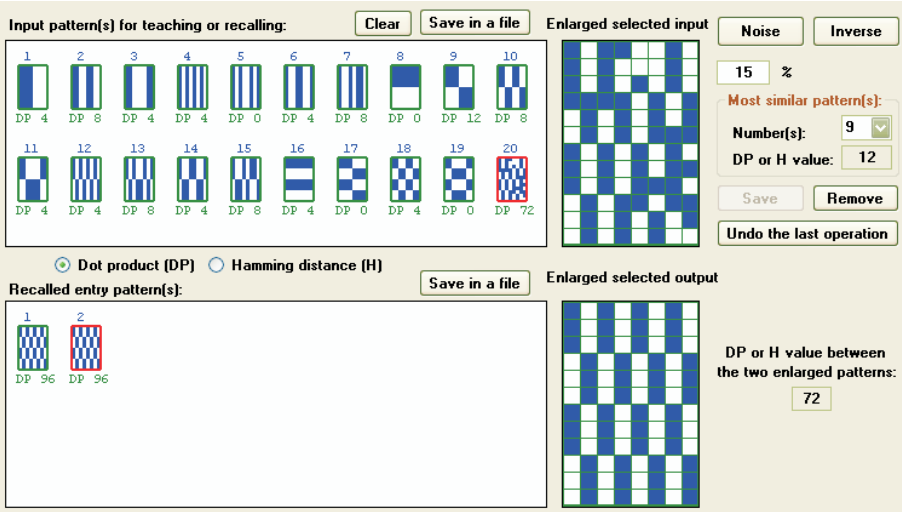



Rys. 11.36. Automatycznie generowany zestaw wzorców pseudolosowych (zapoczątkowany ciągiem znaków X, Y, Z)

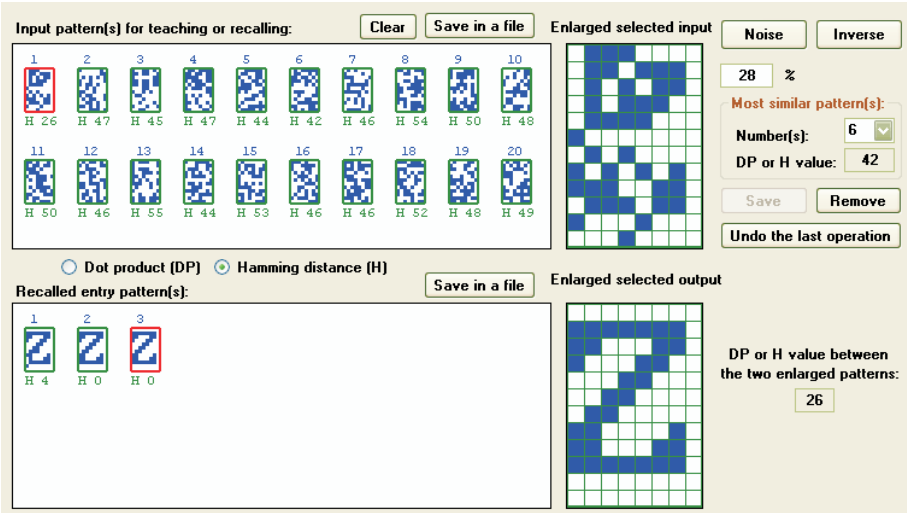
W pierwszym przypadku (to znaczy, gdy każesz generować wzorce od samego początku automatycznie) program zbuduje zestaw wzorców wzajemnie ortogonalnych, co bardzo ułatwi działanie pamięci. Będą one prostsze i bardziej regularne w przypadku w pełni automatycznej generacji wzorców niż w przypadku zainicjowania procesu generacji ręcznie podanym znakiem, bo w tym drugim przypadku generowane są wzorce nie wzajemnie ortogonalne, ale pseudolosowe (por. rys. 11.35 i 11.36). W drugim i trzecim przypadku, tj. gdy przed automatyczną generacją znaków zdecydujesz się wprowadzić jeden lub więcej znaków z klawiatury, program wygeneruje wzorce pseudolosowe. Zauważ małe wartości iloczynów skalarnych.

Ortogonalne wzorce są bardzo pożyteczne przy eksploatacji sieci jako pamięci skojarzeniowej, ponieważ przy korzystaniu z ortogonalnych wzorców (a także wzorców pseudolosowych) możliwe jest odtworzenie oryginalnego obrazu z obrazu wejściowego, nawet mocno zakłóconego (rys. 11.37, 11.38). Dotyczy to oczywiście **każdego** z zapamiętanych wzorców, jednak przy odtwarzaniu wzorców pseudolosowych generowanych przez program dość trudno jest się zorientować, czy naprawdę odtworzony został dokładnie ten sam sygnał, który był podany jako wzorzec przed zniekształceniem – bo wzorce pseudolosowe wyglądają (oględnie mówiąc...) dość egzotycznie.

Zjawisko odtwarzania poprawnych obrazów z bardzo mocno zakłóconych wzorców w sieci Hopfielda zapamiętującej ortogonalne czy pseudolosowe sygnały też ma swoje granice (rys. 11.39, 11.40). Jeśli wzorzec zniekształ-

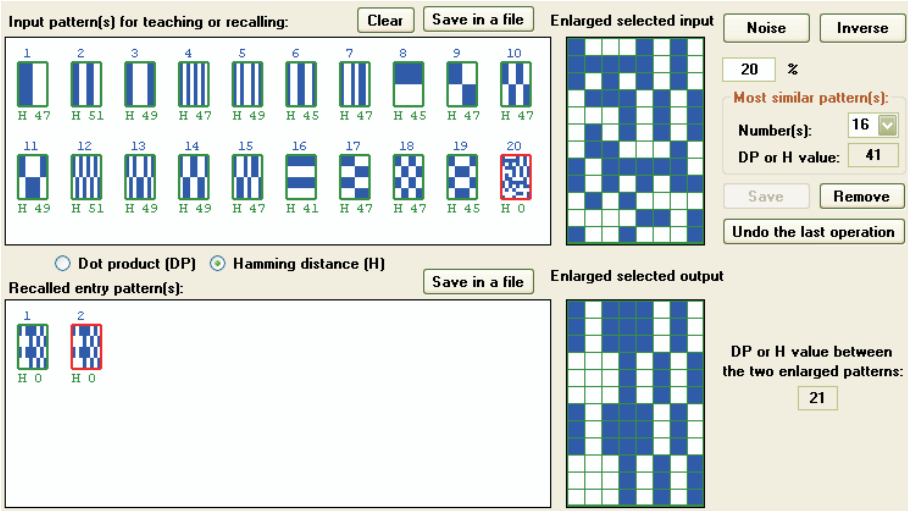



Rys. 11.37. Poprawne odtwarzanie zakłóconego obrazu przy korzystaniu z ortogonalnych wzorców

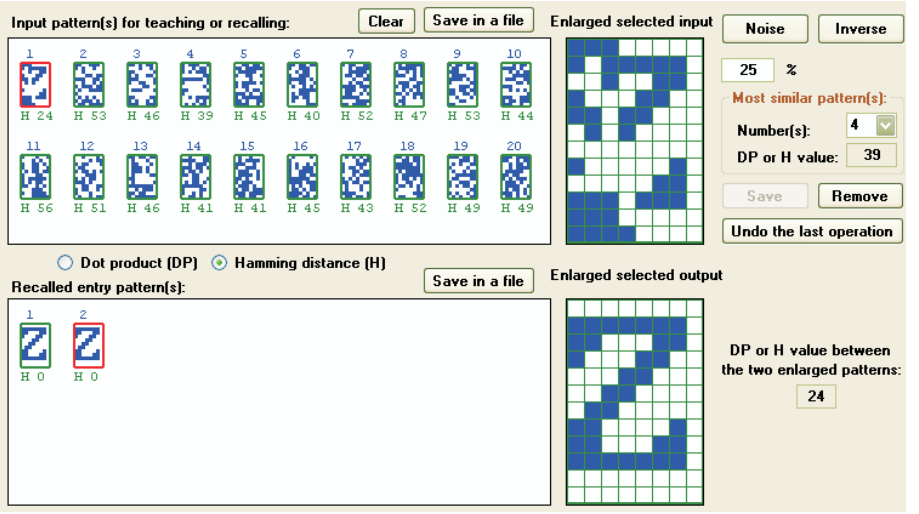



Rys. 11.38. Poprawne odtwarzanie wzorca litery **Z**, w którym zmieniono losowo 26 punktów (tj. zaszumiono 28%)

cisz zbyt mocno – nastąpi bezpowrotna utrata możliwości jego odtworzenia. Zjawisko to ma charakter dość krytyczny. Na przykład na rysunku 11.38 oraz 11.40 pokazałem, że sieć z pseudolosowymi wzorcami potrafi niesłychanie szybko i sprawnie odtworzyć wzorzec litery **Z**, w którym zmieniono losowo 26 oraz 24 punkty.

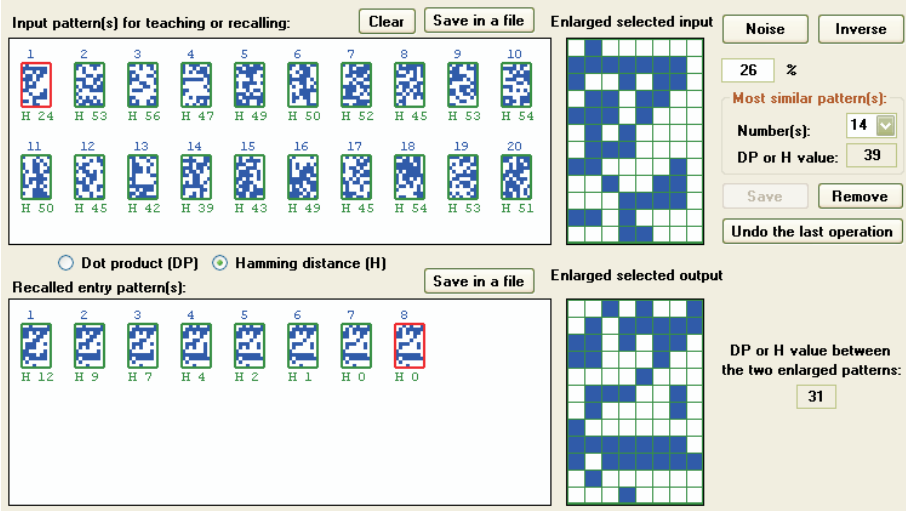



Rys. 11.39. Niedoskonałe odtwarzanie wzorca ortogonalnego, w którym zmieniono losowo 19 punktów (tj. zaszumiono 20%)



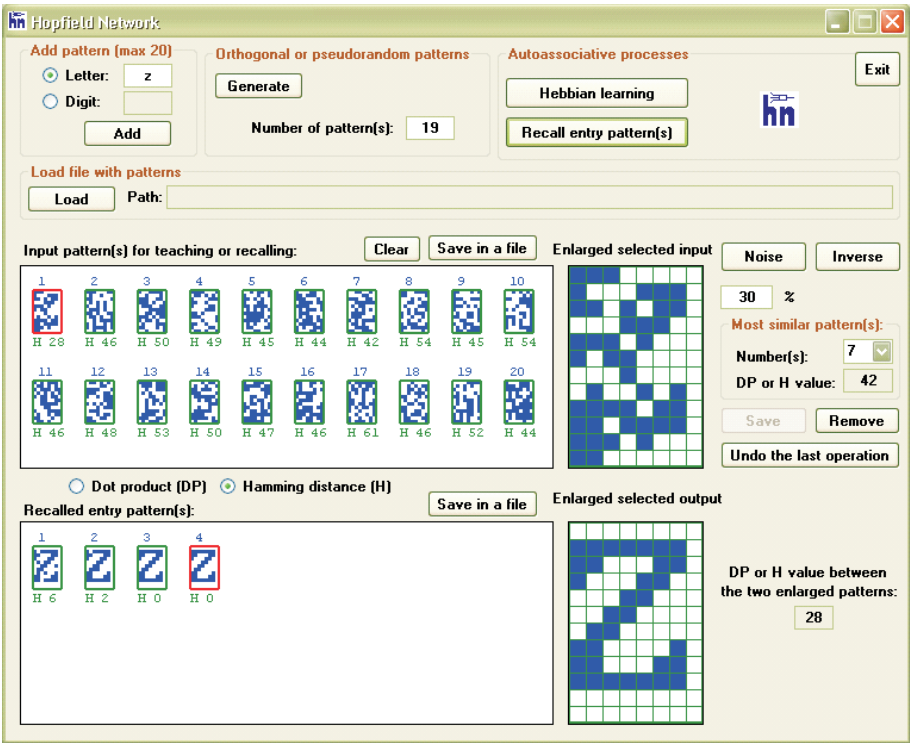

Rys. 11.40. Poprawne odtwarzanie wzorca litery **Z**, w którym zmieniono losowo 24 punkty (tj. zaszumiono 25%)

Możesz powtarzać eksperymenty wiele razy i za każdym razem otrzymasz inny wynik, ponieważ za każdym razem wzorzec zakłócony będzie wyglądał inaczej, a punkty do zamiany wybierane są losowo. Często wystarczy jednak zmiana jednego tylko dalszego punktu (to znaczy zepsucie na obrazie wzorca 25, a nie 24 pikseli) – by oryginalnego obrazu nie dało się poprawnie odtworzyć, gdyż pojawiają się niewielkie, ale nieusuwalne błędy (rys. 11.41).




Rys. 11.41. Całkowicie niepoprawne odtwarzanie wzorca litery **Z**, w którym zmieniono losowo 25 punktów (tj. zaszumiono 26%)

Przy jeszcze większym stopniu destrukcji wejściowego wzorca proces odtwarzania staje się dłuższy, a wynik końcowy jest jeszcze gorszy, aż wreszcie przy jeszcze większej deformacji wzorca ulega totalnej dezintegracji i sieć produkuje na swoim wyjściu już tylko same śmieci) chociaż nawet w tej sytuacji mogą się zdarzać – na zasadzie zwykłego zbiegu okoliczności – niezwykle efektowne przypadki błyskotliwych sukcesów przy odtwarzaniu wzorców bardzo mocno zniekształconych (rys. 11.42).




Rys. 11.42. Incydentalnie poprawne odtwarzanie wzorca litery **Z**

Jeżeli chciałbyś poeksperymentować na wcześniej utworzonym (i zapisanym przez Ciebie na dysku Twojego komputera) zestawie wzorców – możesz to zrobić korzystając z pola grupującego **Load file with patterns**. Przycisk **Load** otwiera okno dialogowe pozwalające na wybranie pliku z wzorcami, który chcesz załadować do programu. Jeżeli ścieżka dostępu do pliku z wzorcami (posiadającego rozszerzenie .PTN) zostanie odnaleziona, program umieści ją w polu edycyjnym **Path:** a miniatury wzorców zawartych we wskazanym pliku pojawią się w oknie graficznym **Input pattern(s) for teaching or recalling:**

## 11.9. Jakie badania można przeprowadzić na pamięci asocjacyjnej?

Za pomocą programu **Example12b** możesz przeprowadzić serię badań, w wyniku których lepiej zrozumiesz, czym jest pamięć asocjacyjna, zbudowana przy użyciu sieci Hopfielda. Wyżej pokazałem Ci już, jak możesz wpisywać informacje do tej pamięci i jak pamięć informacje te odtwarza. Obecnie spróbujemy porozmawiać o tym, jaka jest pojemność pamięci zbudowanej z wykorzystaniem sieci neuronowych.

W normalnej pamięci RAM czy ROM, jaką masz w swoim komputerze, jest konkretny limit pojemności. Wynika to z faktu, że każda wiadomość jest w takiej pamięci zapisywana w oddzielnym miejscu, do którego uzyskujesz dostęp podając (bezpośrednio lub pośrednio) jego adres. Identyczna sytuacja jest na twardym dysku i na dyskach CD. W związku z tym ilość informacji, jaką możesz zapisać w każdej z tych pamięci, jest dokładnie wyznaczona przez liczbę tych zaadresowanych miejsc przeznaczonych do ich przechowywania i dlatego na pytanie, jak dużo miejsca jest w pamięci – można odpowiedzieć szybko i konkretnie.

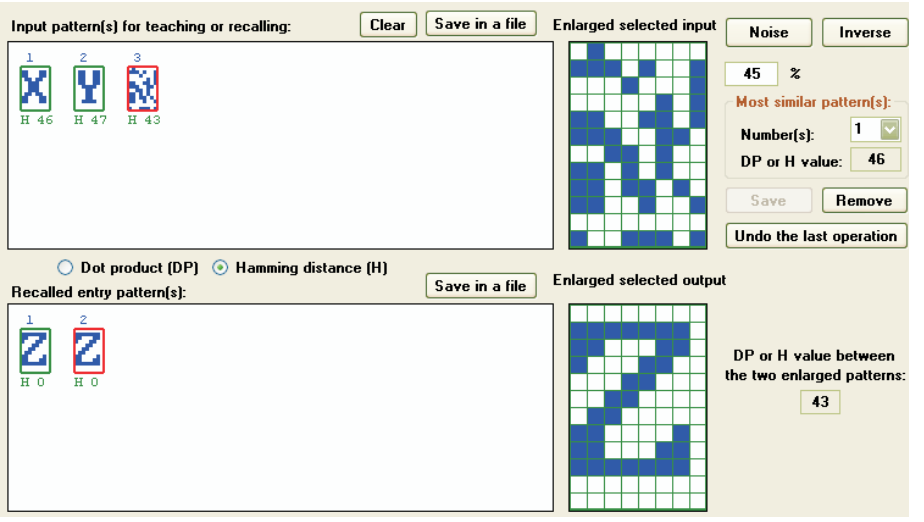
Z pamięcią budowaną w sieci Hopfielda jest inaczej. Tutaj nie ma żadnych oddzielnie zaadresowanych miejsc przeznaczonych dla poszczególnych informacji, gdyż w pamiętaniu (i odtwarzaniu) każdego konkretnego wzorca uczestniczą **wszystkie** neurony sieci, co oznacza, że w poszczególnych neuronach wzorce są nałożone na siebie, co stwarza konkretne problemy (pamiętasz „przesłuchy"?). Co więcej, sposób odtwarzania (odczytywania) informacji z pamięci neuronowej różni się diametralnie od tego, jaki stosują komputery. Zamiast podawać nazwę zmiennej, w której przechowywana jest jakaś wiadomość, albo wskazywać nazwę pliku, który ją zawiera (oba te sposoby w istocie odwołują się do **adresów** wiadomości, których synonimami są nazwy zmiennych lub plików) – w pamięci asocjacyjnej podajesz samą wiadomość jako taką – ale być może niekompletną lub zniekształconą. W związku z tym niesłychanie trudno zorientować się, ile takich wiadomości da się do sieci wepchnąć, a które już się nie zmieszczą.

Jest na ten temat – oczywiście – dokładna i konkretna teoria, która wszystkie te kwestie szczegółowo wyjaśnia. Jeśli kiedyś będziesz jej potrzebował na serio – możesz przeczytać na ten temat w mojej wielokrotnie już tu cytowanej książce pt. *Sieci neuronowe.*

Zadaniem tej książki jest jednak wprowadzenie podstawowych pojęć i koncepcji na temat sieci neuronowych metodą eksperymentowania z nimi. W związku z tym spróbujemy odpowiedzieć na pytanie o pojemność pamięci neuronowej – właśnie eksperymentując z sieciami. Tą drogą będzie także

możliwe dostarczenie Ci maksymalnie ścisłych informacji na ich temat – jednak ponieważ wiadomości te zdobędziesz sam, poprzez samodzielnie prowadzone badania – będziesz je naprawdę rozumiał – a to bardzo ważne.

Pierwszym i najważniejszym czynnikiem decydującym o łatwości i pewności odtwarzania przez sieć zapamiętanych sygnałów – jest liczba danych utrwalanych w sieci w postaci śladów pamięciowych. Przy zapamiętaniu w sieci niewielkiej liczby wzorców (na przykład 3) odtwarzanie informacji jest pewne i niezawodne – nawet przy bardzo dużych zniekształceniach (rys. 11.43).

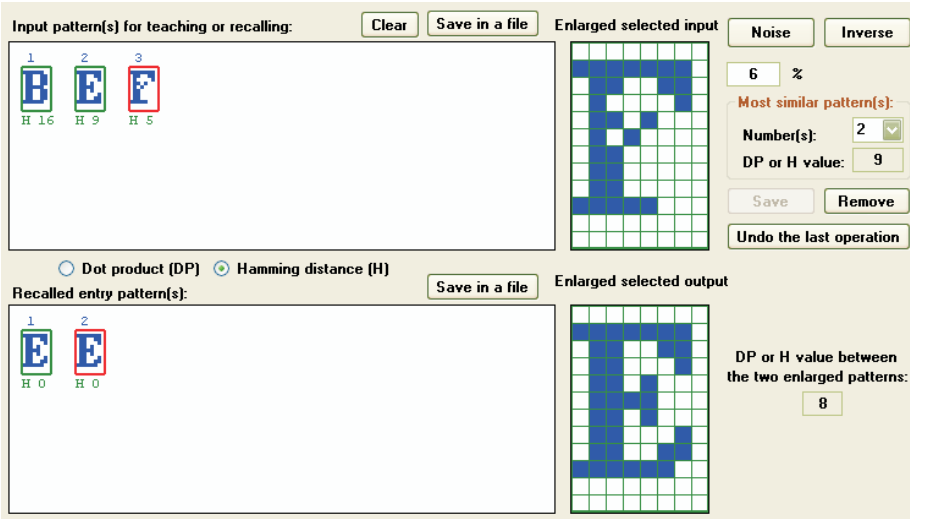



Rys. 11.43. Niezawodne odtwarzanie wiadomości przy niewielkiej liczbie zapamiętanych wzorców

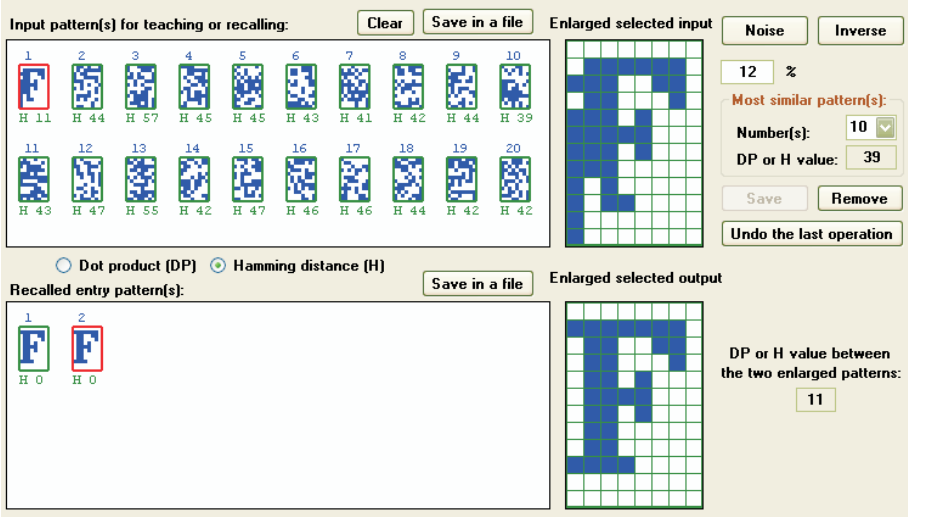
Jednak efekt ten zależny jest także od tego, jak bardzo różnią się od siebie zapamiętywane wiadomości. Na rysunku 11.44 pokazałem Ci wynik eksperymentu, w czasie którego sieć miała do czynienia także z niewielką liczbą wzorców – a jednak na skutek ich dużego podobieństwa nie potrafiła ich poprawnie odtwarzać.

Jak z tego wynika, chcąc zbadać maksymalną pojemność sieci – powinieneś operować sygnałami maksymalnie różniącymi się od siebie – czyli najlepiej sygnałami ortogonalnymi czy pseudolosowymi. Wtedy – jak już wiesz – możesz uzyskiwać dobre wyniki pracy sieci nawet przy maksymalnej sumarycznej liczbie wzorców, wynoszącej w rozważanym tu programie 20 (rys. 11.45)

Przy nie idealnie ortogonalnych wzorcach (np. pseudolosowych, powstających, gdy proces automatycznej generacji rozpoczniesz po podaniu dwóch

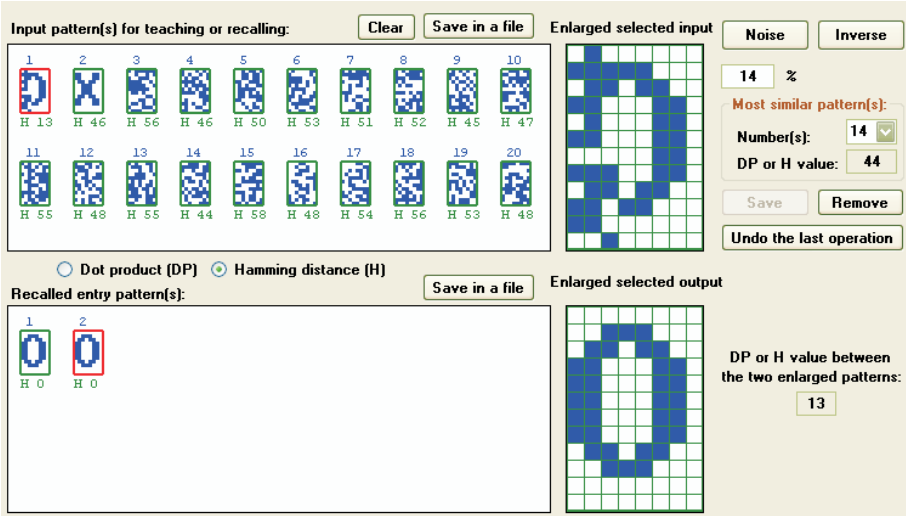



Rys. 11.44. Niepowodzenie przy odtwarzaniu obrazu przy niewielkiej liczbie, ale bardzo podobnych do siebie wzorców

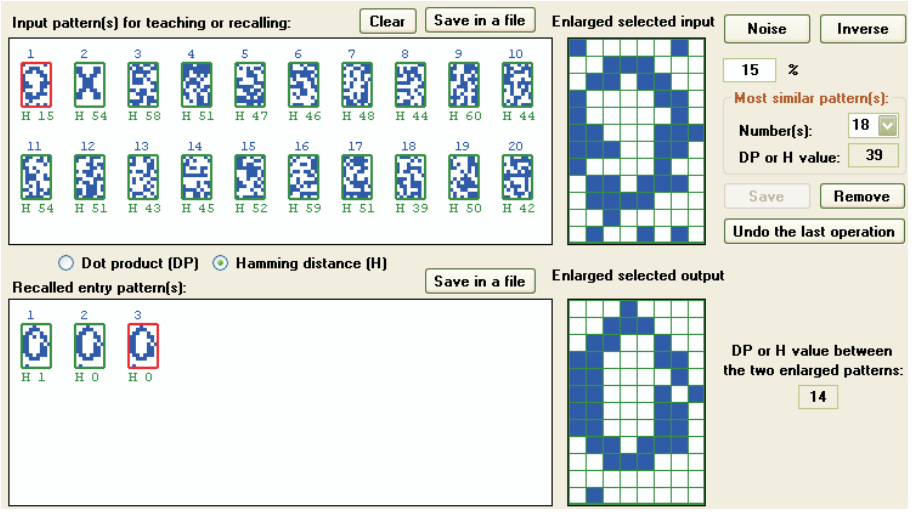



Rys. 11.45. Poprawne odtwarzanie silnie zniekształconego sygnału przy maksymalnej liczbie pseudolosowych wzorców

własnych znaków) – wyniki mogą także być niezłe, pod warunkiem dobrania maksymalnie różniących się od siebie początkowych znaków i przy niezbyt dużych zniekształceniach początkowych obrazów (rys. 11.46), jakkolwiek



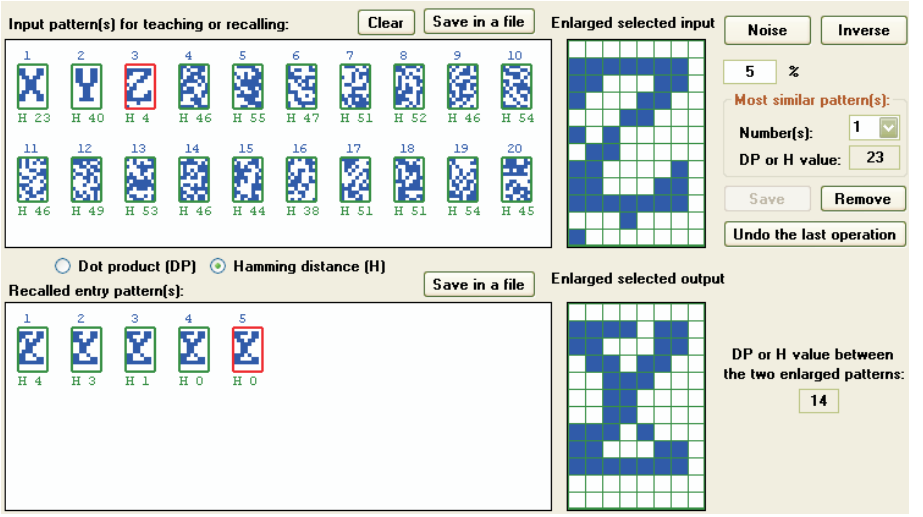

Rys. 11.46. Sukces przy zapamiętywaniu dużej liczby nie całkiem ortogonalnych wzorców




Rys. 11.47. Przesłuchy ujawniające się przy większym poziomie zniekształceń wejściowego sygnału

w tym przypadku użycie bardziej zniekształconego wejścia ujawnia natychmiast istnienie w sieci silnych przesłuchów (rys. 11.47).

Poprawne odtwarzanie wzorca w układzie z wygenerowanymi przez program ortogonalnymi lub prawie ortogonalnymi sygnałami może trochę zacierać fakt, że przy dużej liczbie zapamiętywanych sygnałów trudności




Rys. 11.48. Duże przesłuchy przy minimalnych zniekształceniach wzorca w sieci nie dotrzymującej warunku ortogonalności

odtwarzania obrazów wejściowych generalnie są znacznie większe niż przy odtwarzaniu ich z mniejszej liczby zapamiętanych wzorców. Jednak na rysunku 11.48 możesz zauważyć, że w warunkach, kiedy nieco większa część rozważanych wzorców nie jest ortogonalna – nawet przy ortogonalnych pozostałych wzorcach sieć produkuje naprawdę paskudne przesłuchy nawet przy absolutnie minimalnych zniekształceniach wejściowego sygnału.

## 11.10. Co jeszcze warto zaobserwować w pamięci asocjacyjnej?

Przypominanie sobie przez sieć prawidłowej formy wzorca na podstawie jego zniekształconej karykatury, wprowadzonej do sieci jako sygnał wejściowy, jest procesem dynamicznym bardzo podobnym do tego, jaki oglądałeś w jednoelementowej sieci, przedstawionej w pierwszym omawianym w tym rozdziale programie **Example12a**. Kolejne etapy tego procesu są wyświetlane na ekranie w postaci obrazów, możesz więc wygodnie śledzić, jak zniekształcony obraz w oknie graficznym **Recalled entry pattern(s)** powoli „wyłania się z chaosu". Klikając myszką na dowolnym obrazie i śledząc liczby pokazujące stopień podobieństwa produkowanych przez sieć obrazów do poszczególnych wzorców, możesz obserwować, jak postępuje proces wyłaniania właściwego obrazu.

Jak pamiętasz z wcześniejszego omówienia, program **Example12b** pozwala Ci to badać na dwa sposoby – poprzez iloczyny skalarne i poprzez odległości Hamminga, przy czym zarówno przy wprowadzaniu danych, jak i przy rekonstrukcji obrazu wystarczy naciśnięcie odmiennego przycisku wyboru, by (nie zmieniając niczego innego) zmienić wyświetlanie danych liczbowych w postaci iloczynów skalarnych (**Dot product**, **DP**) na wyświetlanie w postaci odległości Hamminga (**Hamming distance, H**) i odwrotnie. Dzięki opisanym wyżej udogodnieniom możesz obserwować (ilościowo i jakościowo) to, jak kolejno tworzone przez sieć przybliżenia poszukiwanego obrazu zmieniają swoje odległości od poszczególnych zapamiętanych wzorców (lub jak zmieniają się wartości iloczynów skalarnych – ale w tym przypadku interpretacja wyników wymaga już pewnej wiedzy matematycznej). Pozwoli Ci to na wyrobienie sobie poglądu na temat dynamiki procesu odtwarzania wiedzy w pamięciach skojarzeniowych, a także ogólniej – o procesach zmian stanu i wyjść neuronów w sieciach Hopfielda.

Program **Example12b** oferuje Ci także dalsze narzędzia, pozwalające na śledzenie procesu odtwarzania wzorców i na wykrywanie przyczyn takiego lub innego przebiegu tego procesu. Po pierwsze, po prawej stronie okna **Recalled entry pattern(s)** znajduje się okno graficzne **Enlarged selected output**. W nim to pojawia się powiększony obraz wybrany z okna **Recalled entry pattern(s).** Jeżeli zaznaczyłeś myszką jeden wzorzec w oknie **Input pattern(s) for teaching or recalling:** oraz jeden wzorzec w oknie **Recalled entry pattern(s)** – to po prawej stronie okna **Enlarged selected output** w polu edycyjnym: **DP or H value between the two enlarged patterns:** pojawi się wartość miary podobieństwa pomiędzy tymi dwoma wzorcami (wyrażona za pomocą iloczynu skalarnego lub odległości Hamminga). Śledząc, jaką ewolucję przebywa wydobywany prze sieć obraz, można więc nie tylko widzieć, co sieć robi, ale także odpowiedzieć sobie na pytanie dlaczego tak się dzieje, ale to wymaga już sporej cierpliwości i dużej wiedzy, ponieważ proces poszukiwania przez sieć właściwego śladu pamięciowego kończy się, gdy kolejna iteracja nie wnosi do obrazu już żadnych poprawek (sieć osiągnęła stan ustalony). Zatrzymanie możliwe jest też w przypadku (bardzo rzadko występującym w praktyce) pojawienia się oscylacji. Jeśli kolejne kroki przechodzą przez te same sekwencje stanów – proces też się zatrzyma, chociaż wynik może być wtedy dość dziwny.

Bardzo ciekawe są obserwowane niekiedy momenty „przeskoku" – kiedy sieć zamiast doskonalić odtwarzanie prawidłowego wzorca nagle skojarzy sobie zdeformowany obraz z zupełnie innym wzorcem i właśnie ten fałszywy wzorzec zaczyna pracowicie czyścić i polerować (rys. 11.32). Często w następstwie takiej przygody sieć tworzy sobie zupełnie nowe wyobrażenie

nie istniejącego wzorca, będącego hybrydą istniejących elementów, ale zestawionych w sposób niezwykły i nie mający nic wspólnego z rzeczywistością (rys. 11.27). Powstają takie twory z bajki lub sennego koszmaru – jak węże ze skrzydłami czy psy o trzech głowach – tylko zestawiane z elementów świata, który był sieci dostępny – czyli kawałków obrazów liter... To trzeba zobaczyć, żeby zrozumieć, jak bardzo odmienne jest działanie sieci neuronowej w stosunku do tępego i pozbawionego fantazji działania zwykłych algorytmów!

Resztę odkryjesz już sam podczas pracy z programem. Powodzenia!

## 11.11. Pytania, zadania i problemy do samodzielnego rozwiązania

1. Spróbuj krótko (maksymalnie w dziesięciu zdaniach) opisać wszystkie różnice, jakie zachodzą pomiędzy siecią neuronową nie posiadającą sprzężeń zwrotnych (taką sieć określa się często angielskim terminem *feedforward*) oraz siecią posiadającą sprzężenia zwrotne.
2. Spróbuj znaleźć formułę, pozwalającą przewidywać, jaka będzie ustalona wartość sygnału wyjściowego w sieci symulowanej przez program **Example12a** przy różnych wartościach parametrów (współczynników wag synaptycznych) i przy różnych wartościach sygnału wejściowego do sieci. Wyniki swoich przemyśleń skonfrontuj z symulacjami uzyskiwanymi za pomocą programu.
3. Przeczytaj jakiś artykuł ze strony internetowej albo książkę na temat *teorii chaosu.* Jak sądzisz, jaki związek ma ta teoria i opisywane w niej zjawiska z dziedziną rekurencyjnych sieci neuronowych?
4. Obejrzyj w jakiejś książce albo w Internecie pochodzące z mózgu człowieka zapisy zmiennych w czasie potencjałów elektrycznych, tak zwanych elektroencefalogramów (angielski skrót EEG). Jak sądzisz, czy przebieg tych sygnałów wskazuje na brak sprzężeń zwrotnych w mózgu człowieka – czy wprost przeciwnie?
5. Spróbuj zastanowić się, czy w przypadku pamięci człowieka (która ponad wszelką wątpliwość jest pamięcią asocjacyjną) miewamy do czynienia ze zjawiskami analogicznymi do „przesłuchów" pojawiających się w sieci Hopfielda. Jeśli tak, to w jaki sposób są one subiektywnie odczuwane i oceniane przez samych ludzi?
6. Na podstawie przeprowadzonych eksperymentów spróbuj ustalić swój własny pogląd, która ze stosowanych miar podobieństwa (**DP** albo **H**) pozwala skuteczniej przewidywać, jakie wzorce będą dobrze odtwarzane przez sieć, a które będą sprawiały trudności.

7. Pomiędzy liczbą neuronów w sieci Hopfielda a liczbą wzorców, które można w takiej sieci skutecznie zapamiętać (z minimalnym prawdopodobieństwem wystąpienia „przesłuchów"), istnieje zależność, dająca się wyprowadzić w sposób matematyczny. Nie wchodząc w szczegóły, można powiedzieć, że im więcej neuronów liczy sieć, tym więcej wzorców można w niej zapamiętać, chociaż każdy poszczególny wzorzec angażuje **wszystkie** neurony całej sieci. Wiedząc o tym spróbuj zastanowić się nad możliwościami pamięciowymi ludzkiego mózgu, który składa się z około stu miliardów neuronów. Czy można uzasadnić fakt, że jedni ludzie mają lepszą, a inni gorszą pamięć?

8. Sieci Hopfielda omówione w tej książce były wszystkie bez wyjątku sieciami **auto-asocjacyjnymi**, to znaczy pamięć skojarzeniowa w nich użyta służyła do odtwarzania zapamiętanych wzorców – i do niczego więcej. Zastanów się, jaki schemat powinna mieć sieć, w której możliwa byłaby realizacja sieci hetero-asocjacyjnej, to znaczy zapamiętującej skojarzenia różnych wzorców (na przykład rysunku przedstawiającego w uproszczeniu jakiś przedmiot oraz litery, od której zaczyna się nazwa tego przedmiotu).

9. Przy badaniu sieci Hopfielda posługiwaliśmy się bardzo prostymi wzorcami, będącymi w istocie wyłącznie uproszczonymi zarysami liter i cyfr. Niewątpliwie znacznie ciekawsze byłoby używanie sieci, która by mogła operować obrazkami podobnymi do przedstawionych na rysunku 11.13. Spróbuj wyjaśnić, dlaczego tego nie zrobiono.

10. Przy budowie dużych sieci Hopfielda (złożonych z dużej liczby neuronów) pewien zasób komputerowy wyczerpuje się szczególnie szybko. Zastanów się, jaki i dlaczego właśnie ten.

11. **Zadanie dla zaawansowanych.** Zbuduj wersję programu **Example12a**, w której zamiast tabeli wartości wyjściowych, produkowanych na wyjściu sieci w kolejnych krokach symulacji, pokazywany będzie rysunek pokazujący czasowy przebieg sygnału wyjściowego w formie graficznej (jako wykres). Jaką trudność jest przy tym najtrudniej pokonać i w jaki sposób poradziłeś sobie z tym problemem?

12. **Zadanie dla zaawansowanych.** Zbuduj wersję programu **Example12b**, w której macierz neuronów, reprezentująca poszczególne zapamiętywane obrazy, będzie miała znacznie większe rozmiary (na przykład będzie powiększona trzykrotnie w każdym kierunku). Przy użyciu takiego udoskonalonego narzędzia przeprowadź podobne doświadczenia, jak opisane w treści rozdziału, a uzyskane obserwacje spróbuj skonfrontować ze stwierdzeniem podanym w pytaniu nr 7.