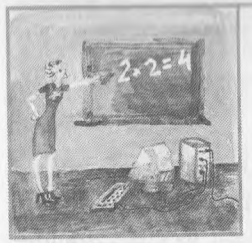
dzeń innych metod może być trudność w podaniu jawnej definicji funkcji przystosowania, jak to miało miejsce w iteracyjnym dylemacie więźnia. Innym powodem mogłaby być po prostu złożoność problemu. W rozdziale 10 opisaliśmy NP-zupełny problem — problem zbyt skomplikowany na to, by można go było rozwiązać za pomocą komputera w rozsądnym czasie. Można zastosować algorytmy genetyczne do znalezienia wprawdzie nie najlepszych, ale jednak dobrych rozwiązań takich problemów. Szczególnie przydatne okazały się w planowaniu i zarządzaniu; na tym polu są one szeroko stosowane w praktyce.

Alternatywą do ewolucyjnego budowania zakodowanego rozwiązania danego problemu jest budowanie kompletnego programu do rozwiązywania problemu. Metoda ta nosi nazwę **programowania genetycznego**. Idea genetycznego programowania jest prosta, lecz jej praktyczna realizacja wymaga pokonywania wielu trudności takich, jak wymyślenie sposobu mutacji programu, który tylko troszeczkę zmieniałby jego funkcjonalność. Innym ważnym zastosowaniem symulowanej ewolucji jest ewolucja *sieci neuronowych*. Sieci neuronowe opiszemy dokładnie w następnym rozdziale.

13



Komputer się uczy

Sieci neuronowe

Pisząc w pierwszym rozdziale, że modelem mózgu ludzkiego jest komputer, posłużyliśmy się ogromnym skrótem myślowym. **Mózg ludzki** jest tak złożony, ma tak wiele aspektów i działa na tak wielu poziomach, że ludzkość jest wciąż bardzo daleko od zrozumienia mechanizmu jego działania. Możemy więc tylko modelować niektóre aspekty jego działań i to w bardzo ograniczonym zakresie.

Jednym z takich uproszczonych modeli jest koncepcja zaproponowana przez von Neumanna, która stanowi podstawę wszystkich nowoczesnych komputerów. Jej dwa główne składniki — centralny procesor (CPU) i pamięć — to odpowiedniki dwóch głównych czynności naszego mózgu — myślenia i zapamiętywania. Bardzo uproszczony model naszego procesu myślowego wykonuje CPU; jest to cykliczne powtarzanie następujących rozkazów.

- Pobierz z pamięci instrukcję do wykonania.
- Pobierz z pamięci dane potrzebne do wykonania instrukcji.
- Wykonaj instrukcję.
- Zapisz uzyskane dane w pamięci.

Mało znanym faktem jest to, że pierwszy cyfrowy programowalny komputer (Z3) został zbudowany w 1941 roku przez Konrada Zuse przed von Neumannem i od niego niezależnie. Zuse zbudował swój pierwszy komputer (Z1) w salonie swoich rodziców w 1938 roku. Komputer Z3 miał wszystkie podstawowe cechy współczesnych komputerów; był programowalny i stosował arytmetykę zmiennoprze-

cinkową. Jednakże przełomowe osiągnięcie Zusa odkryto i doceniono znacznie później.

Konrad Zuse (1910-1995) – niemiecki inżynier i wyprzedzający swoje czasy pionier komputeryzacji. Zaplanował i zbudował pierwszy komputer i rozwinął pierwszy wysokiej klasy język programowania – Plankalkul

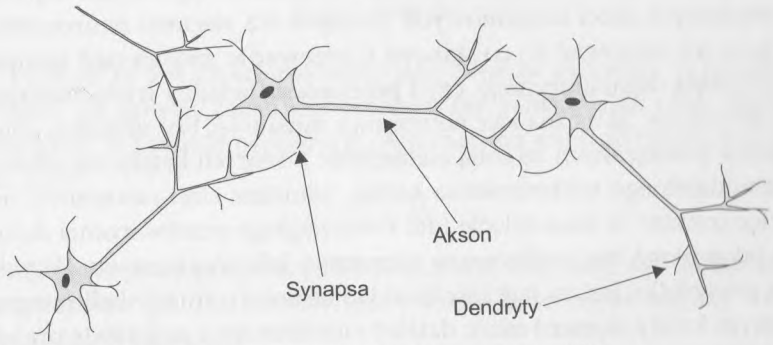
Nie musimy chyba przekonywać Czytelników, jak przydatny okazał się ten model. Zakres zastosowań współczesnych komputerów mówi za siebie. Jest jednak całkiem jasne, że „mózg" komputera to coś zupełnie innego niż nasz mózg. Istotną wadą komputerów opartych na koncepcji von Neumanna jest ich brak odporności na błędy. Nawet mała zmiana w zbiorze instrukcji dla CPU (w programie komputerowym) zwykle powoduje, że program taki przestaje w ogóle działać. Drugim ograniczeniem jest sekwencyjna natura komputerowego procesu – to, że procesor może wykonywać w danej chwili tylko jedną czynność. Jeszcze jedną wadą komputera jest jego niemożność dostosowywania się do zmieniających się okoliczności. Lekarstwem na wszystkie te problemy jest alternatywny model komputerowy — **sieć neuronowa**.

Sieci neuronowe (a właściwie *sztuczne sieci neuronowe*) są też modelami mózgu, ale starają się naśladować również jego strukturę. Mózg jest siecią wielu aktywnych elementów (komórek) powiązanych ze sobą gęstą siecią połączeń; komórki te działają równolegle i komunikują się ze sobą. Innymi słowy, jest to **układ rozproszony**. Układy rozproszone są na ogół bardziej odporne na błędy niż układy scentralizowane. Układ scentralizowany ma zawsze słaby punkt — swoje centrum, którego zniszczenie czyni nieprzydatnym cały układ. Układy rozproszone nie mają takiego jednego słabego punktu i często mogą wciąż działać, mimo iż jakaś ich część uległa zepsuciu lub uszkodzeniu.

Taka idea przyświecała przed 30 laty twórcom eksperymentalnej sieci komputerowej ARPAnet. Nazwa pochodzi od Agencji Zaawansowanych Projektów Badawczych (Advanced Research Projects Agency), która finansowała ten projekt. Ta pierwsza sieć została zbudowana przez Ministerstwo Obrony USA i ze względu na swoje rozproszenie miała być odporna na różne nawet poważne zakłócenia (spowodowane na przykład przez jądrowy atak ze strony ZSRR). Nikt się nie spodziewał, że ARPAnet przekształci się w ogromną światową sieć zwaną Internetem i że głównym zagrożeniem dla jej prawidłowego działania będzie nie zewnętrzny wróg, a raczej autorzy wirusów komputerowych i pocztowe spamy. Rozproszona natura tej sieci doskonale spełniła swoje oczekiwania; wędrujące w tej sieci pakiety trafiają do odbiorców nawet wtedy, gdy wiele odcinków sieci nie działa z powodu wirusa.

Imitacja mózgu

Mózg ludzki (podobnie jak zwierzęcy) składa się z wyspecjalizowanych komórek zwanych **neuronami**. Neurony są zdolne do przekazywania informacji między sobą. Poza typowymi elementami każdej komórki są one dodatkowo wyposażone w dendryty i w akson (porównaj rys. 13.1). Dendryty zbierają sygnały z sąsiednich neuronów. Gdy całkowite natężenie zebranych sygnałów przekracza pewien próg, wzdłuż aksonu jest wysyłany impuls elektryczny (zwany **potencjałem aktywacji**). Gdy tylko sygnał dociera do końca aksonu, może zostać przejęty przez neurony znajdujące się w pobliżu tego końca. Aksony niektórych neuronów są krótkie, lecz niektóre mogą być bardzo długie.



Rys. 13.1. Schematyczny rysunek neuronów powiązanych między sobą

Sygnały nie przechodzą bezpośrednio z jednego neuronu do drugiego. Każde połączenie neuronów jest w istocie maleńkim zaworem zwanym **synapsą**. Niektóre synapsy przekazują rzeczywiście elektryczne sygnały między neuronami, lecz większość przepuszcza substancje chemiczne zwane neurotransmiterami. Potencjał aktywacji uruchamia końcówkę aksonu do wysłania neurotrasmitera w synapsę. Ten zaś rozprzestrzenia się i wiąże z receptorami odbierającego neuronu, tym samym przekazując informację.

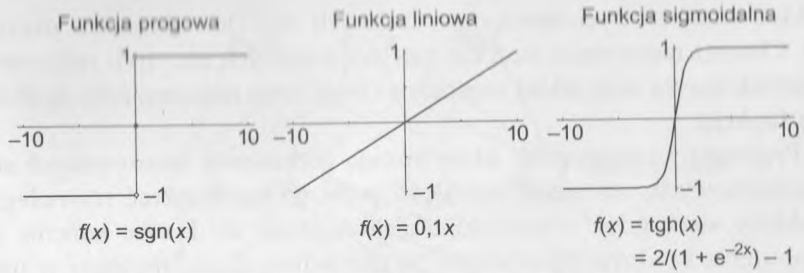
Kiedy sygnał przechodzi przez synapsę, może się wzmacniać lub osłabiać w zależności od własności synapsy. Te własności (**wagi**) synaps mają wpływ na sposób, w jaki mózg przetwarza informację. Z drugiej strony procesy zachodzące w mózgu mogą spowodować zmiany wagi synapsy. Te dwa fakty doprowadziły kanadyjskiego psychologa Donalda Hebba do hipotezy, że regulowanie wag synaps jest podstawą zapamiętywania i uczenia się. Hipoteza ta, nosząca nazwę **teorii Hebba**, została sformułowana w 1949 roku i stała się podwaliną psychologii poznawczej.

Pojedynczy neuron jest bardzo powolny w porównaniu z procesorem nowoczesnego komputera. Reakcja na impuls zabiera mu około milisekundy (10^{-3} s), podczas gdy 1 GHz procesor wykonuje jedną instrukcję w nanosekundę (10^{-9} s). Jednakże, gdy weźmiemy pod uwagę, że mózg ludzki składa się z grubsza z 10^{11} równocześnie działających neuronów i że każdy neuron jest połączony z tysiącami (a czasami nawet z setkami tysięcy) innych neuronów, to możemy śmiało powiedzieć, że mózg ma o wiele rzędów wielkości większą moc obliczeniową niż jakikolwiek zbudowany dotąd komputer.

Przedstawiony powyżej schematyczny opis działania ludzkiego mózgu jest bardzo uproszczony. Jest to w istocie jeszcze jeden model, który ujmuje zaledwie kilka aspektów tego procesu, opuszczając wiele innych. Ale ten prosty model wystarcza już do zrozumienia ogólnej zasady budowy **sztucznych sieci neuronowych** zwanych też sieciami neuronowymi. Sieć taka ma imitować to, co stanowi o przewadze mózgu nad komputerem — pełną demokratyzację, czyli przekazanie władzy wielu rozproszonym ośrodkom. Sztuczna sieć neuronowa musi więc być układem zbudowanym z powiązanych ze sobą elementów, z których każdy ma zdolność do samodzielnego wykonywania zadań. Sztuczne sieci neuronowe można więc uważać za rozwinięcie idei równoległego przetwarzania danych. Sieci takie mogą być realizowane sprzętowo lub programowo. W pierwszym przypadku jest to najczęściej układ scalony o dużej skali integracji, w którym każdy element może działać niezależnie na podstawie przychodzących do niego sygnałów. Kilka firm produkuje takie fizyczne realizacje sieci neuronowych w postaci układu zwykłych pojedynczych procesorów. Ze względu jednak na brak elastyczności takich układów popyt na nie jest niewielki. W drugim przypadku żadnej fizycznej sieci nie ma, a cała sieć neuronowa działa na jednym zwykłym komputerze i jest tylko *modelowana* przez odpowiedni program. Takie sieci cieszą się dużo większą popularnością. Budując je na wzór mózgu ludzkiego będziemy używali tej samej terminologii.

Każde sztuczne złącze neuronów, odpowiednik synapsy, ma przypisaną sobie liczbową wagę; gdy sygnał (który też jest liczbą) przechodzi przez złącze, zostaje pomnożony przez tę wagę. Sztuczny neuron przejmuje nadchodzące sygnały, stosuje pewne proste funkcje do ich sumy i wysyła wynik.

Funkcję, której używa sztuczny neuron do przekształcenia sumy przychodzących sygnałów w sygnał wysyłany, nazywamy **funkcją aktywacji**. Jak już wspomnieliśmy wcześniej, aktywacja prawdziwego neuronu następuje skokowo przy przekroczeniu pewnej granicy sumarycznego natężenia wszystkich przychodzących sygnałów. Modelowanie takiego zachowania oznacza zastosowanie **progowej** funkcji aktywacji (patrz rys. 13.2)



Rys. 13.2. Trzy rodzaje funkcji aktywacji stosowanych w sieciach neuronowych

w następującej ogólnej postaci:

$$f(x) = a\,\mathrm{sgn}(x) + c.$$

Funkcja ta jest często używana jako funkcja aktywacji w sztucznych sieciach. Jej główną zaletą jest prostota, ma jednak kilka wad, takich jak nieciągłość w punkcie progowym. Alternatywą jest funkcja liniowa (patrz rys. 13.2)

$$f(x) = ax + c.$$

Jednak najczęściej stosowanymi jako funkcje aktywacji są funkcje **sigmoidalne** (patrz rys. 13.2) stanowiące swoisty kompromis między dwiema poprzednimi funkcjami

$$f(x) = \frac{a}{1 + e^{-bx}} + c.$$

Najbardziej pospolitymi funkcjami sigmoidalnymi są funkcje najprostsze, np. $f(x) = 1/(1 + e^{-x})$, albo funkcja tangens hiperboliczny $f(x) = 2/(1 + e^{-2x}) - 1 = \mathrm{tgh}\,(x)$.

Często wiąże się z każdym neuronem dodatkowy parametr zwany **wagą przesunięcia**. Wagę przesunięcia dodaje się do sumy przychodzących sygnałów przed zastosowaniem funkcji aktywacji, w rezultacie przesuwając wykres tej funkcji w prawo lub w lewo i przesuwając próg (jeśli taki istnieje).

Wśród neuronów w mózgu wyróżniamy neurony czuciowe, ruchowe i pośredniczące. Neurony pośredniczące przekazują informacje od jednego neuronu do drugiego, jak to było wcześniej opisane. Neurony czuciowe odbierają informacje od bodźców zewnętrznych, a nie od innych neuronów. Z kolei neurony ruchowe przekazują informacje do mięśni powodując ich skurcze. Czuciowe i ruchowe neurony umożliwiają dwustronne oddziaływanie mózgu ze środowiskiem. Podobnie, sztuczne sieci neuronowe mają układ neuronów wejściowych, układ neuronów wyjściowych

i układ wewnętrznych neuronów (zwanych zwykle neuronami ukrytymi). Chociaż najprostsze sieci nie zawierają żadnych ukrytych neuronów, to jednak muszą mieć jakieś wejściowe i wyjściowe neurony, żeby spełniać swą funkcję.

Programy komputerowe, które imitują zachowanie rzeczywistych sieci neuronowych, nie mogą jednak w pełni wykorzystywać równoległej struktury sieci, gdyż większość z nich realizuje się koniec końców na komputerze z jednym procesorem. Są one jednak dużo łatwiejsze w użyciu niż sieci sprzętowe i są szeroko stosowane wtedy, gdy zdolność adaptacji i tolerancja błędów bardziej niż równoległość jest argumentem za wyborem takiej metody rozwiązywania problemu.

Gdy próbujemy rozwiązać problem, stosując sieć neuronową, napotykamy dwie główne trudności.

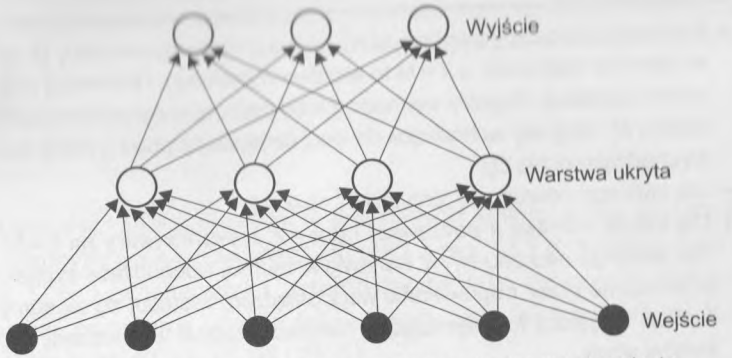
- **Projektowanie sieci** — decyzja, ile wybrać neuronów i jak określić powiązania między nimi; innymi słowy, konstrukcja skierowanego grafu reprezentującego sieć.
- **Uczenie sieci** — dopasowanie wag na podstawie jakichś próbnych danych wejściowych tak, by sieć mogła „uczyć się" spełniać swoje zadanie.

Wciąż jeszcze nie mamy pojęcia, jak przyroda rozwiązuje te problemy. Wydaje się, że plan naszego mózgu jest w dużej mierze wynikiem ewolucji, choć połączenia między neuronami na pewno zmieniają się w ciągu naszego życia. Istnieje parę teorii mówiących o tym, jak różne bodźce mogą wpływać na wagi synaps, prowadząc do uczenia się, ale nikt nie rozumie, jak dokładnie przebiega ten proces. Nie możemy być nawet pewni tego, że wagi synaps rzeczywiście są podstawą naszej pamięci.

Nie ma też uniwersalnej metody budowania sztucznych sieci neuronowych i ich uczenia. W ogólności każdy skierowany graf może służyć jako projekt sieci. Duży stopień dowolności pociąga za sobą trudność w formułowaniu szczegółowych twierdzeń na temat sieci. Najczęściej wprowadza się dodatkowe ograniczenia i twierdzenia są formułowane dla pewnej węższej rodziny sieci. W dalszym ciągu tego rozdziału opiszemy kilka najpopularniejszych rodzin sieci neuronowych i związane z nimi algorytmy.

Perceptrony

Najprostszym i najpopularniejszym rodzajem sieci neuronowych jest **wielowarstwowy perceptron** (MLP). Należy on do sieci niezawierających żadnych zamkniętych pętli (cykli). Informacja dostarczona do neuronów wejściowych przechodzi przez taką sieć raz zanim dojdzie do neuronów wyj-



Rys. 13.3. Sieć neuronowa z warstwą wejściową, wyjściową i jedną warstwą ukrytą

ściowych i wtedy stan sieci się stabilizuje. MLP jest zbudowany jako ciąg warstw: warstwa wejściowa, pewna liczba warstw ukrytych i warstwa wyjściowa. Każdy neuron w warstwie połączony jest z każdym neuronem w sąsiedniej warstwie. Połączenia te zawsze są skierowane od warstwy wejściowej do warstwy wyjściowej. Zobacz na rys. 13.3, jak wygląda schemat budowy dwuwarstwowego perceptronu. (Niektórzy nazwaliby tę sieć trójwarstwowym perceptronem, na ogół jednak nie liczy się warstwy ukrytej).

Sieć MLP przetwarza sygnały wprowadzone do warstwy wejściowej na sygnały odbierane w warstwie wyjściowej według algorytmu z rys. 13.4.

Dla lepszego zrozumienia tej procedury i związanej z nią notacji rozważmy prosty dwuwarstwowy perceptron z rys. 13.5. Sieć taka w przypadku dwu sygnałów wchodzących danych liczbami 0 lub 1 daje na wyjściu funkcję logiczną *wyłączne lub* tych dwóch wartości. Funkcja wyłączne lub, oznaczana symbolem XOR, jest zdefiniowana następującym wzorem

$$\mathrm{XOR}(x,y) = \begin{cases} 0, \text{ gdy } x = y \\ 1, \text{ gdy } x \neq y. \end{cases}$$

Dwóch pionierów sztucznej inteligencji Marvin Minsky i Seymour Papert w swojej pracy z 1969 roku zatytułowanej *Perceptrony* udowodnili, że żaden perceptron bez ukrytych neuronów nie jest w stanie obliczyć funkcji XOR. Wyrazili też przypuszczenie, że to samo dotyczy też bardziej skomplikowanych perceptronów. Ostudziło to na jakiś czas zainteresowanie wszystkimi sieciami neuronowymi. Na szczęście podany wyżej przykład potwierdza, że sieci MLP nie są wcale tak bezużyteczne, jak myśleli Minsky i Papert.

Marvin Minsky (1927-) – pionier sztucznej inteligencji; wynalazł między innymi konfokalny mikroskop skanujący i żółwia Logo (razem z Papertem).

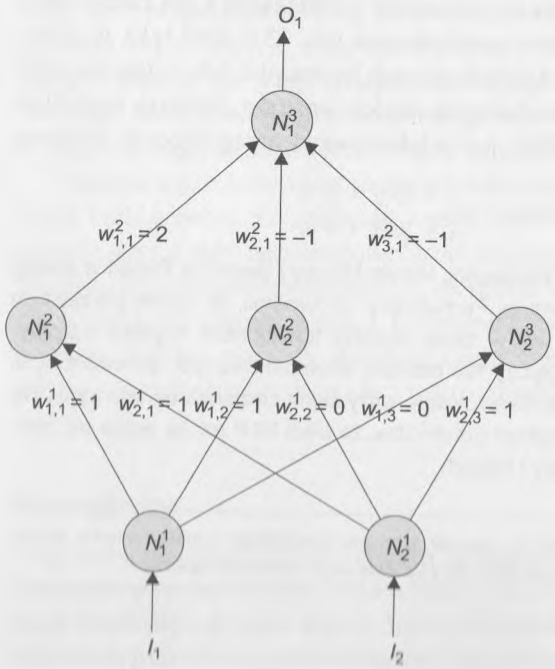
- Neurony oznaczamy symbolami N_i^k, gdzie k numeruje warstwy ($k = 1$ to warstwa wejściowa, a $k = M$ to warstwa wyjściowa), i numeruje neurony w warstwie. Sygnały wychodzące od neuronów oznaczamy symbolami S_i^k. Sygnały wchodzące do sieci oznaczamy przez I_i natomiast wychodzące przez O_i.
- Dla każdego neuronu N_i^1 przyjmij S_i^1 równy I_i.
- Dla każdej warstwy k począwszy od $k = 2$, a skończywszy na $k = M$: Dla każdego neuronu N_i^k w warstwie k zsumuj dochodzące sygnały pomnożone przez odpowiednie wagi połączeń, zastosuj do tej sumy funkcje aktywacji f, otrzymując w rezultacie sygnał wychodzący S_i^k. Innymi słowy,

$$S_i^k = f\left(\sum_{j=1}^{L_{k-1}} w_{j,i}^{k-1} S_j^{k-1}\right),$$

gdzie L_{k-1} jest liczbą neuronów w warstwie $k-1$, a $w_{j,i}^{k-1}$ jest wagą połączenia neuronu N_j^{k-1} z neuronem N_i^k.

- Dla każdego neuronu w warstwie wyjściowej ($k = M$) sygnał S_i^M jest i-tą składową sygnału wychodzącego z sieci O_i.

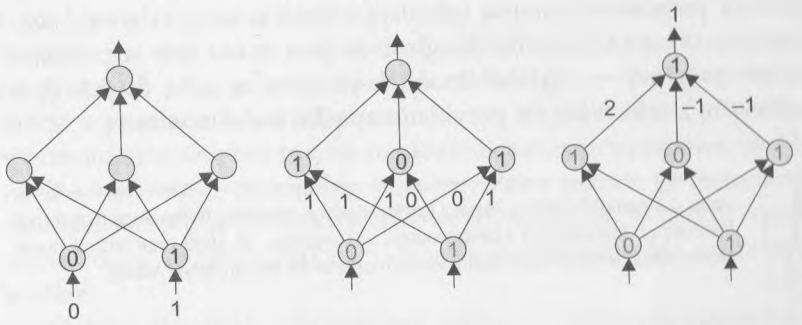
Rys. 13.4. Algorytm stosowany przez $(M-1)$-warstwowy perceptron do przekształcania wektora sygnałów wejściowych o L_1 składowych $\{I_1, I_2, ..., I_{L_1}\}$ w wektor sygnałów wyjściowych o L_M składowych $\{O_1, O_2, ..., O_{L_M}\}$



Rys. 13.5. Prosty MLP, który oblicza funkcję XOR dla dwóch liczb binarnych. Funkcją aktywacji jest $f(x) = \text{sgn}(x)$

Seymour Papert (1928-) – matematyk amerykański; twórca języka programowania Logo.

Na rysunku 13.6 pokazano, w jaki sposób kolejne kroki podanego powyżej algorytmu zastosowanego do sieci z rys. 13.5 prowadzą do wyniku XOR(0, 1) = 1. Zachęcamy Czytelników, by jako ćwiczenie wykazali, że sieć ta daje poprawne wyniki także wtedy, gdy jako sygnały wchodzące weźmiemy pary (0, 0), (1, 0) i (1, 1).



$S_1^1 = I_1 = 0$
$S_2^1 = I_2 = 0$

$S_1^2 = \text{sgn}(1\times0 + 1\times1) = 1$
$S_2^2 = \text{sgn}(1\times0 + 0\times1) = 0$
$S_3^2 = \text{sgn}(0\times0 + 1\times1) = 1$

$O_1 = S_1^3 =$
$= \text{sgn}(2\times1 - 1\times0 - 1\times1) = 1$

Rys. 13.6. Obliczenie wykonane przez MLP z rys. 13.5. Sieć obliczyła, że XOR(0, 1) = 1

Sieć z rysunku 13.5 umożliwia w bardzo skomplikowany sposób obliczyć funkcję XOR za pomocą komputera. To samo można w końcu uzyskać za pomocą jednej elementarnej instrukcji dla procesora. Celem omawiania tego przykładu było zapoznanie się Czytelników z sieciami neuronowymi i związaną z nimi terminologią. Nie udało nam się jednak pokazać, do czego sieci neuronowe mogą się naprawdę przydać. Udało nam się zaplanować topologię sieci i wagi połączeń odpowiednie dla postawionego celu. Nie było to łatwe, a w rezultacie otrzymaliśmy bardzo skomplikowaną metodę obliczania bardzo prostej funkcji. Nie wykorzystaliśmy głównej cechy sieci neuronowych, a mianowicie ich zdolności *uczenia się*.

Uczenie sieci

Uczenie sieci neuronowych polega na dopasowywaniu ich wag połączeń tak, by sieć działała w pożądany przez nas sposób. W ostatnim przykła-

dzie dobraliśmy te wagi arbitralnie w pewnym celu (aby obliczyć funkcję XOR dwóch binarnych liczb). Takie podejście może być dobre w przypadku niektórych sprzętowych sieci neuronowych, ale na ogół jest bardzo mało wydajne. Dla rozwiązania zadania dużo łatwiej jest zaplanować algorytm zamiast tworzyć sieć. Mówiąc o procesie uczenia sieci neuronowej, mamy zwykle na myśli zastosowanie jakiegoś **algorytmu uczenia**, który dopasowywałby *automatycznie* wagi połączeń w sieci. Są różne takie algorytmy odpowiednie dla różnych zadań i dla różnych topologii sieci.

Najprostszy chyba algorytm uczenia pochodzi od Johna Hopfielda i opiera się na obserwacjach Hebba dotyczących prawdziwych procesów zachodzących w mózgu. Hebb zauważył, że gdy często się zdarza, że dwa neurony połączone synapsą odpalają równocześnie, to wtedy waga tej synapsy wzrasta. Teza Hebba głosi, że proces ten leży u podstaw **pamięci asocjacyjnej** — czyli zdolności kojarzenia ze sobą dwóch (lub więcej obiektów), takich jak na przykład zapachu bzów z wiosną.

John J. Hopfield (1933-) – fizyk z Kalifornijskiego Instytutu Technologii (California Institute of Technology) i Uniwersytetu w Princeton. W 1982 przedstawił nowe sformułowanie teorii sieci neuronowych oparte na fizyce statystycznej.

Najprostsza sieć Hopfielda działa jak **pamięć autoasocjacyjna**, co oznacza, że kojarzy ona obiekt z nim samym. Przy danym układzie sygnałów wchodzących produkuje ten sam układ jako sygnały wychodzące. Wydaje się to bezcelowe; zauważmy jednak, że sieci neuronowe zdają się być odporne na błędy. Gdy więc wprowadzimy zniekształconą nieco wersję zapamiętanego układu, sieć wyprodukuje mimo wszystko oryginalny zapamiętany układ. Można, na przykład, uczyć sieć z układem liter zakodowanych w postaci bitmapowych (złożonych z białych i czarnych kropek) obrazów. Później, przy skanowaniu tekstu można przepuścić każdą skanowaną literę przez sieć, aby usunąć błędy powstałe w procesie skanowania. Można by uzyskać to samo, porównując każdą skanowaną literę z literami z układu, a następnie zastępując ją najbardziej do niej podobną literą z tego układu. Sieci neuronowe realizowane sprzętowo rozwiązują jednak ten problem znacznie szybciej.

Prosta sieć Hopfielda do rozpoznawania binarnych wzorów (zapisanych jako ciągi złożone z -1 i $+1$) o długości n to jednowarstwowy perceptron z n neuronami wchodzącymi i z n neuronami wychodzącymi. Funkcją aktywacji jest $f(x) = \mathrm{sgn}(x)$.

Będziemy używali terminu **zbiór treningowy** dla określenia przykładowych danych używanych w procesie uczenia. W tym przypadku zbiorem treningowym jest zbiór wzorców do zapamiętywania $\{P^1, P^2, \ldots, P^m\}$, gdzie k-ty wzorzec jest ciągiem $P^k = \{p_1^k, p_2^k, \ldots, p_n^k\}$ liczb $+1$ i -1. Uczenia sieci na tym zbiorze dokonuje się przez przyjęcie wag połączeń w postaci

$$w_{i,j} = \frac{1}{n} \sum_{k=1}^{m} p_i^k p_j^k.$$

Tę prostą regułę interpretujemy w następujący sposób. Dla każdego wzorca w zbiorze (P^k) i dla każdego połączenia (między neuronem wchodzącym i a neuronem wychodzącym j) dodaj do wagi połączenia człon $1/n$, jeśli zachodzi dodatnia korelacja między liczbami w ciągu na pozycjach i i j ($p_i^k = +1$ i $p_j^k = +1$ lub $p_i^k = -1$ i $p_j^k = -1$) i odejmij od wagi połączenia ten sam człon, jeśli zachodzi ujemna korelacja między liczbami na pozycjach i i j ($p_i^k = +1$ i $p_j^k = -1$ lub $p_i^k = -1$ i $p_j^k = +1$). Informacja o korelacjach jest więc zapisana w wagach sieci i gdy następnie wprowadzimy do sieci trochę zniekształcone dane wejściowe, sieć będzie zdolna odtworzyć oryginalny wzorzec. Żeby jednak ta metoda dobrze działała, wzorce ze zbioru treningowego muszą się w istotny sposób od siebie różnić, a zniekształcenie danych wejściowych nie może być zbyt wielkie.

Metoda Hopfielda uczenia jest prosta — dostarcza gotowych wzorów dla wszystkich wag połączeń — ale jej zastosowania są ograniczone. Może jedynie spowodować, że sygnały wychodzące z sieci są identyczne z wchodzącymi, filtrując czasem po drodze jakieś szumy. Co jednak zrobić, jeśli chcemy, by sieć obliczyła coś nowego?

Piętnaście lat po wynalezieniu sieci neuronowych amerykański psycholog Frank Rosenblatt wymyślił pierwszy algorytm do ich uczenia. Opisał on najprostszą formę perceptronu (warstwa wejściowa, warstwa wyjściowa i funkcja sgn jako funkcja aktywacji) i symulował ją na komputerze IBM 704. Opisał też algorytm uczenia sieci zwany **regułą uczenia perceptronu** (PLR).

PLR stosuje zbiór treningowy w postaci par złożonych z sygnału wejściowego i pożądanego sygnału wyjściowego. Na początku przyjmuje się losowe wartości wszystkich wag w sieci. Następnie dokonuje się serii iteracji. Za każdym razem zadaje się jako sygnały wejściowe dla sieci sygnały ze zbioru treningowego $\{I_1, I_2, \ldots, I_n\}$. Sieć produkuje jakieś sygnały wyjściowe $\{O_1, O_2, \ldots, O_m\}$ i te sygnały odejmuje się od pożądanych sygnałów wyjściowych $\{D_1, D_2, \ldots, D_m\}$, otrzymując wektor błędu $\{E_1, E_2, \ldots, E_m\}$. Długość wektora błędu jest miarą tego, jak dobrze działa sieć. Jeśli wynosi zero, oznacza to, że sieć produkuje dokładnie pożądany wynik. Im dłuższy wektor błędu, tym gorsze zachowanie sieci. Każda waga połączeń w sieci jest stopniowo dopasowywana w zależno-

ści od tej składowej wektora błędu, za którą odpowiada. Tak więc, jeśli $E_j = 0$, to $w_{i,j}$ pozostaje niezmieniona, lecz gdy $E_j = \pm 1$, to modyfikuje się $w_{i,j}$. Wzrost lub zmniejszenie wagi zależy od małego parametru zwanego **tempem uczenia** (zwyczajowo oznaczanym symbolem η) i jest też proporcjonalny do sygnału wejściowego I_i. Tak wiec zmodyfikowana waga jest dana wzorem $w_{i,j} + \eta I_i E_j$. Tempo uczenia powinno być jakąś małą dodatnią liczbą, na przykład 0,1. Jeśli tempo uczenia jest za duże, to każda iteracja powoduje, że sieć „zapomina", czego nauczyła się w czasie poprzednich iteracji. Gdy zaś tempo uczenia jest za małe, wtedy proces uczenia trwa bardzo długo. Algorytm PLR jest przedstawiony szczegółowo na rys. 13.7.

1. Wybierz losowo wartości wszystkich wag.
2. Dopasuj wagi:
   a) Przyjmij $E_{\text{total}} = 0$.
   b) Dopasuj wagi zgodnie z jakimś wzorcem ze zbioru treningowego:
      (i) Weź następny wektor sygnałów wejściowych $\{I_1, I_2, ..., I_n\}$ i odpowiadający mu wektor pożądanych sygnałów wyjściowych $\{D_1, D_2, ..., D_m\}$ ze zbioru treningowego.
      (ii) Zastosuj algorytm dla perceptronu z rys. 13.4, by otrzymać z sygnałów wejściowych $\{I_1, I_2 ..., I_n\}$ sygnały wyjściowe $\{O_1, O_2, ..., O_m\}$.
      (iii) Odejmij ostatnie sygnały wyjściowe od pożądanych sygnałów wyjściowych, by otrzymać wektor błędu, to znaczy $E_i = D_i - O_i$ dla $i = 1, 2, ..., m$.
      (iv) Dodaj kwadrat długości wektora błędu do całkowitego błędu $E_{\text{total}} = E_{\text{total}} + (E_1^2 + E_2^2 + ... + E_O^2)$.
      (v) Każdą wagę połączeń i-tego neuronu z warstwy wchodzącej z j-tym neuronem warstwy wychodzącej, $w_{i,j}$, skoryguj według wzoru
      $$w_{i,j} = w_{i,j} + \eta I_i E_j,$$
      gdzie η jest tempem uczenia.
   c) Jeśli są jeszcze jakieś wzorce w zbiorze treningowym, wróć do punktu 2b.
3. Jeśli całkowity błąd E_{total} jest mniejszy niż poprzednim razem, wróć do punktu 2.

Rys. 13.7. Reguła uczenia perceptronu

Wielokrotne iterowanie tego algorytmu prowadzi do zmniejszenia całkowitego błędu sieci... do pewnego punktu. W idealnej sytuacji w punkcie tym całkowity błąd wynosi zero, a więc sieć produkuje dokładnie pożądane sygnały wyjściowe dla każdego wzorca ze zbioru treningowego. Problem w tym, że taka nauczona sieć powinna zachowywać się w pożądany

przez nas sposób również wtedy, gdy wprowadzimy do niej wzorce inne niż te ze zbioru treningowego. Czy tak jest w istocie, zależy od rodzaju zagadnienia, które próbujemy rozwiązać. Proste perceptrony są wystarczająco dobre dla wielu zastosowań. Choć zdolne są do nauczenia tylko prostych funkcji, z drugiej strony mogą jednak operować na funkcjach o dowolnej liczbie argumentów.

Rozważmy zagadnienie optycznego rozpoznawania liter (OCR). Szukamy funkcji, która przekształcałaby obraz bitmapowy w pojedynczą literę. Z matematycznego punktu widzenia taka funkcja nie jest skomplikowana. Jest jednak funkcją bardzo wielu argumentów — tylu, ile pikseli zawiera bitmapa. Człowiekowi byłoby bardzo trudno określić, jak każdy poszczególny piksel wpływa na otrzymaną w rezultacie literę, a zatem, jak zaprogramować komputer, by rozpoznawał litery.

PLR dostarcza sposobu nauczenia komputera rozpoznawania liter przez dostarczanie mu zbioru przykładowych bitmap — par liter. W żadnym momencie nie musimy w ogóle znać *mechanizmu* przekształcania bitmapy w litery — sieć może to sama „wykombinować", oczywiście z pewnym błędem.

Bernard Widrow i Marcian E. Hoff zaproponowali ogólniejszy sposób uczenia perceptronów, a mianowicie **regułę delta** znaną też jako **reguła Widrowa-Hoffa**. Opiera się ona na metodzie **najszybszego spadku**.

Metoda najszybszego spadku jest sposobem minimalizowania błędu, gdy błąd ten zależy od wielu czynników. Występuje to właśnie w przypadku sieci neuronowych; sieć produkuje pewien błąd, gdy działa na sygnały wejściowe ze zbioru treningowego, a błąd ten zależy od wszystkich wag w sieci. Naszym celem jest znalezienie najniższej wartości funkcji błędu. Trudność wynika z dużej liczby zmiennych — wag w sieci. Metoda najszybszego spadku każe nam zacząć w przypadkowym miejscu (odpowiada to losowym wagom początkowym) i posuwać się wzdłuż każdej z osi wykresu w kierunku opadającego zbocza. Aby sobie unaocznić ten proces, wyobraź sobie, że stoisz na zboczu wzgórza. W kierunku północnym i zachodnim zbocze się wznosi, a w kierunku południowym i wschodnim opada, przy tym w kierunku wschodnim dwa razy bardziej stromo niż w południowym. W tym przypadku metoda najszybszego spadku każe zrobić jeden krok na południe i dwa kroki na wschód, jeśli oczywiście chcesz znaleźć się na dole.

Zastosowanie metody najszybszego spadku dla korekty zachowania jednowarstwowego perceptronu oznacza obliczenie dla każdej wagi pochodnej cząstkowej błędu po tej wadze, a następnie poprawienie wagi proporcjonalnie do tej pochodnej. Pochodna cząstkowa wyznacza kierunek i spadzistość stoku na wykresie błędu wzdłuż osi odpowiadającej tej wadze. Innymi słowy, mówi nam, w którym kierunku i jak silnie ta właśnie waga wpływa na błąd. Korekta wag jednowarstwowego perceptronu z funkcją aktywacji $f(x)$ według metody najszybszego spadku jest dana następującym wzorem:

$$w_{i,j} = w_{i,j} + \eta f'\left(\sum_{k=1}^{n} w_{k,j} I_k\right) I_i (D_j - O_j),$$

gdzie η jest tempem uczenia. Wzór ten różni się od poprzedniej reguły uczenia perceptronu pojawieniem się pochodnej funkcji aktywacji $f'(\sum w_{k,j} I_k)$. Wyraz ten to prawie sygnał wyjściowy z j-tego neuronu wyjściowego, lecz zamiast funkcji aktywacji występuje pochodna funkcji aktywacji obliczona dla sumy ważonych sygnałów wejściowych. Metoda najszybszego spadku wymaga, by funkcja aktywacji była różniczkowalna, co nie zachodzi w przypadku nieciągłej funkcji $f(x) = \text{sgn}(x)$. W przypadku liniowej funkcji aktywacji pochodna jest stałą i otrzymujemy taki sam wynik, jak poprzednio.

Jak wykazali Minsky i Papert, jednowarstwowe perceptrony są z natury ograniczone w swych możliwościach. Nie mogą nawet obliczyć prostej funkcji XOR dwóch wartości logicznych! Wielowarstwowe perceptrony są dużo potężniejsze. Pozostaje jednak pytanie, jak uczyć wielowarstwowe sieci.

Najczęściej stosowanym algorytmem do uczenia wielowarstwowych perceptronów, i pewnie wszystkich sieci neuronowych, jest **propagacja wsteczna**. Propagacja wsteczna przypomina regułę uczenia perceptronów w tym, że zaczyna od losowych wag i stosuje zbiór par sygnałów wchodzących i pożądanych sygnałów wychodzących do stopniowego poprawiania tych wag. Metoda ta nazwana jest propagacją wsteczną, gdyż najpierw koryguje się wagi prowadzące do warstwy wychodzącej, potem wagi poprzedzające je, i tak dalej, aż do osiągnięcia dolnej warstwy. Kolejność poprawiania wag jest przeciwna do kolejności, w jakiej są obliczane sygnały, gdy sieć wykonuje swoje zadanie.

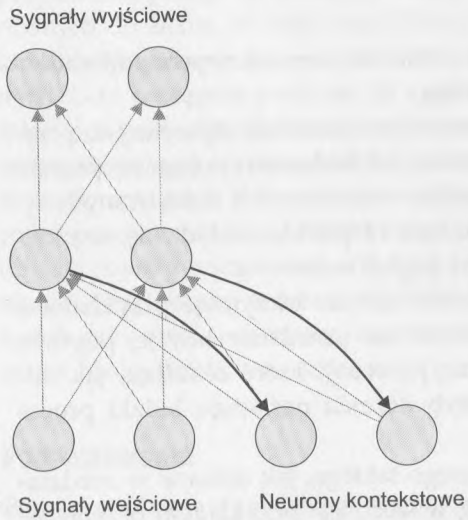
Tak jak i reguła delta, propagacja wsteczna stosuje do błędu sieci metodę najszybszego spadku opisaną już wcześniej w tym rozdziale. Wagi połączeń najbliższe sygnałów wyjściowych oblicza się podobnie jak w przypadku jednowarstwowego perceptronu. Trudniej jest obliczyć poprawki do wag położonych głębiej w sieci, gdyż wpływają one na wszystkie wyjściowe neurony, a nie tylko na jeden z nich. Na szczęście okazuje się, że można użyć częściowych wyników otrzymanych przy korygowaniu jednej warstwy do znacznego uproszczenia obliczeń dla warstwy ją poprzedzającej — stąd bierze się wsteczny kierunek algorytmu.

Stosując program **Hopfield**, możesz zbudować sieć neuronową i nauczyć ją rozpoznawania twojego charakteru pisma. Możesz zbudować zbiór treningowy składający się z przykładowych liter, wybrać plan sieci i uczyć ją na tym treningowym zbiorze. Nauczona sieć jest zdolna rozpoznawać napisane przez ciebie litery, mimo iż nigdy nie piszesz dokładnie tak samo. Sieć jest wielowarstwowym perceptronem z funkcją aktywacji sigmoid i obciążonymi wagami. Algorytm do nauki stosuje propagację wsteczną z pędem. Dodatkowe informacje można znaleźć w opisie programów.

Sieci rekurencyjne

Jak już wspomnieliśmy wcześniej, perceptrony należą do sieci jednokierunkowych, niezawierających żadnych zamkniętych pętli. Sygnał przechodzi przez taką sieć raz — od neuronów wchodzących do wychodzących. **Rekurencyjne** sieci zawierają pętle tak, że informacja może być przekazywana między neuronami w obu kierunkach. Cecha ta czyni sieci rekurencyjnie potencjalnie potężniejszymi, ale powoduje też, że trudniej znaleźć dla nich algorytm.

Najprostszą siecią rekurencyjną jest **sieć Elmana**, nazwana tak od swego wynalazcy Jeffa Elmana. Sieć Elmana jest dwuwarstwowym perceptronem z dodatkowymi neuronami zwanymi **neuronami kontekstowymi**. Liczba neuronów kontekstowych jest równa liczbie neuronów ukrytych. Umieszczone są one na poziomie warstwy wchodzącej i połączone ze wszystkimi ukrytymi neuronami tak, jakby były dodatkowymi wchodzącymi neuronami. Dodatkowe połączenia (o wadze jeden) dające **sprzężenie zwrotne** prowadzą od neuronów z warstwy ukrytej do neuronów kontekstowych (patrz rys. 13.8). Za każdym razem, gdy sieć przetwarza jakiś sygnał, stan ukrytej warstwy jest przechowywany w neuronach kontekstowych. Stan ten jest wprowadzany następnie wraz z sygnałem wchodzącym, gdy sieć znów coś przetwarza.



Rys. 13.8. Prosta sieć Elmana z dodatkowymi połączeniami zaznaczonymi pogrubionymi liniami

W odróżnieniu od sieci jednokierunkowych, które zawsze produkują ten sam sygnał wyjściowy przy tym samym sygnale wejściowym, sygnał wychodzący z sieci rekurencyjnej zależy nie tylko od aktualnego sygnału wchodzącego, ale też od poprzednich sygnałów wejściowych. Cecha ta

jest przydatna, gdy operujemy danymi przychodzącymi seriami, jak na przykład dane z giełdy. Przypuśćmy, że sieć neuronowa ma na celu przewidywanie notowań zamknięcia jakiejś firmy w danym dniu. Jako sygnał wejściowy przyjmuje ona notowania otwarcia, wartości innych pokrewnych akcji i, powiedzmy, kurs dolara. Jeśli jest to standardowa jednokierunkowa sieć, to oprze swoje przewidywanie na tych danych. Jeśli jednak jest to sieć Elmana, to weźmie też pod uwagę informacje z poprzedniego dnia. Sposób, w jaki to robi, zależy też od treningu. Najprostszą wersję sieci Elmana można uczyć przy użyciu standardowego algorytmu propagacji wstecznej, gdyż wagi sprzężeń zwrotnych są ustalone i nie wymagają modyfikacji.

Jak już wspomnieliśmy, prosta sieć Hopfielda może występować też w potężniejszej rekurencyjnej postaci. Algorytm uczenia rekurencyjnej sieci Hopfielda jest wciąż prosty, jeśli ustalamy wartości wszystkich wag równocześnie według prostego wzoru. Jednakże zamiast przesyłać sygnał jednokrotnie przez sieć, przesyłamy go wielokrotnie, stosując za każdym razem sygnał wyjściowy jako sygnał wejściowy w następnym przebiegu. Proces ten zbiega w końcu do punktu, w którym sygnał wyjściowy pokrywa się z jednym z zapamiętanych wzorców, a więc nie zmienia się już w trakcie przetwarzania przez sieć.

Różne rodzaje uczenia

Istnieją trzy typy algorytmów do uczenia sieci neuronowych: **pod nadzorem**, **ze wzmacnianiem** i **bez nadzoru**.

Wszystkie dotychczas omawiane w tym rozdziale algorytmy to przykłady uczenia pod nadzorem. Uczenie pod nadzorem polega na prezentowaniu sieci przykładowych sygnałów wejściowych i oczekiwanych sygnałów wyjściowych. W przypadku sieci Hopfielda zakłada się, że oczekiwanym sygnałem wyjściowym jest sygnał wejściowy.

Podczas uczenia ze wzmocnieniem sieć też otrzymuje przykładowe sygnały wejściowe. Tym razem jednak nie przedstawiamy jej poprawnych odpowiedzi, lecz tylko dajemy jej oceny, które określają, jak dobrze przetwarza te sygnały. Ten tryb uczenia naśladuje ludzki proces uczenia się.

Stosowanie algorytmu ewolucyjnego takiego, jak opisany w rozdziale 13, do znajdowania wartości wag w sieci, jest przykładem uczenia ze wzmacnianiem. Zwykły algorytm genetyczny do trenowania sieci polega na zakodowaniu wszystkich wag w genotypie, a następnie zastosowaniu standardowej techniki ewolucyjnej dla znalezienia właściwego zbioru wag. To podejście wymaga możliwości porównywania sieci (rozumianej jako zbiór wag) z innymi sieciami, by zapewnić jej genetyczną przewagę, która odpowiada ocenianiu sieci pod względem jakości sygnałów wyjściowych.

Ewolucyjny algorytm rzadko jest używany do uczenia sieci, ze względu na dostępność innych algorytmów, takich jak na przykład metoda najszybszego spadku. Znacznie ważniejsze zastosowanie znajduje przy projektowaniu sieci neuronowych. Wspomnieliśmy o kilku algorytmach dla znajdowania wag połączeń w sieci, lecz pominęliśmy problem wyboru topologii sieci (na przykład liczby ukrytych warstw w perceptronie). Prawdą jest, że nie ma na to ścisłych metod. Poza pewnymi empirycznymi twierdzeniami na ten temat (jak to, że na ogół wystarcza jedna warstwa ukryta), do każdej reguły istnieją wyjątki. Algorytm ewolucyjny okazuje się bardzo dobrym narzędziem dla optymalizacji topologii sieci neuronowych.

Przy uczeniu bez nadzoru przedstawiamy sieci jakieś przykładowe sygnały wejściowe i nie dostarczamy żadnych dodatkowych informacji. Może wydać się dziwnym, jak bez nadzoru można nauczyć sieć, by działała tak, jak byśmy chcieli. W żadnym momencie nie precyzujemy przecież, czego chcemy. O to właśnie chodzi; czasami sami nawet nie wiemy, do czego dążymy, analizując zbiór danych. Sieć neuronowa może nam wtedy pomóc w szukaniu jakiejś prawidłowości i znajdowaniu sposobu klasyfikacji danych. Taka klasyfikacja danych nazywa się **grupowaniem**. Proces grupowania polega na dzieleniu zbioru obiektów na grupy obiektów podobnych do siebie. W 1981 roku Teuvo Kohonen opisał sieć neuronową, która dzięki uczeniu bez nadzoru bada wewnętrzną strukturę zbioru danych. Gdy następnie poda się jej sygnał wejściowy, sieć wskazuje, jakie miejsce zajmuje ten sygnał w strukturze. Taki rodzaj sieci nosi nazwę **samoorganizującej się mapy** (SOM) lub **mapy Kohonena**. Sieci te nazywamy mapami, gdyż obrazują one wielowymiarowe dane na niskowymiarowej przestrzeni — najczęściej na płaszczyźnie. Wielowymiarowe dane opisują obiekty o wielu cechach. Mogą to być układy dźwięków, optyczne obrazy, ciągi tekstowe lub inne obiekty. Sieci SOM uczą się umieszczać każdy taki obiekt na niskowymiarowej siatce tak, by podobne obiekty leżały blisko siebie, a te, które różnią się bardziej, były dalej.

Podsumowanie

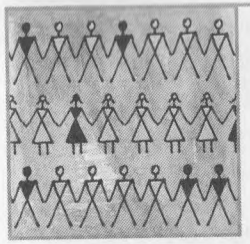
Pomysł sieci neuronowych jest starszy nawet od idei komputera von Neumanna i przez wszystkie te lata przeżywał wiele wzlotów i upadków. W 1943 roku Warren McCulloch i Walter Pitts opisali sztuczny neuron i fantazjowali na temat sieci takich neuronów, ale niewiele mogli zdziałać przy ówczesnym stanie technologii. W późnych latach pięćdziesiątych Frank Rosenblatt symulował perceptron na jednym z pierwszych

komputerów, wzbudzając wielkie zainteresowanie sieciami neuronowymi. To zainteresowanie osłabło w końcu lat sześćdziesiątych po tym, jak Minsky i Papert stwierdzili istnienie ograniczeń dotyczących perceptronu, by znów odrodzić się w latach osiemdziesiątych z pojawieniem się komputerów osobistych i rozwojem teorii sieci neuronowych (dzięki pracom Hopfielda). Obecnie sieci neuronowe znajdują szerokie zastosowanie w ekonomii, finansach, medycynie, naukach ścisłych, sporcie i wielu innych dziedzinach ludzkiej działalności. Wiele wyspecjalizowanych firm na całym świecie zajmuje się produkcją oprogramowania wykorzystującego sieci dla różnych celów. Podajemy poniżej przykładowe zastosowania sieci neuronowych znalezione na stronach internetowych tych firm:

- przewidywanie kursów walut,
- prognozy giełdowe,
- ocena ryzyka inwestycji,
- ocena reakcji rynku,
- selekcja policjantów z uwagi na skłonność do nieetycznego zachowania,
- powoływanie przysięgłych,
- przewidywanie napraw drogowych,
- diagnozy lekarskie,
- przewidywanie wyników wyścigów konnych,
- przewidywanie wybuchów na słońcu,
- prognozy pogody,
- sprawdzanie jakości produktów,
- synteza mowy,
- rozpoznawanie mowy,
- rozpoznawanie pisma,
- sprawdzanie podpisu,
- automatyczne sterowanie,
- wybór składników karmy dla kurcząt,
- poszukiwanie złota,
- badanie zanieczyszczeń powietrza,

i to jeszcze nie wyczerpuje listy. Niektórzy używają nawet sieci neuronowych do przewidywania wygranej na loterii. Choć ze zrozumiałych względów w tej dziedzinie sieci nie są przydatne, to w większości innych dają bardzo dobre wyniki. Uczeni wciąż rozwijają teorię sieci neuronowych. Tymczasem możliwość zastosowania jej w praktyce (w projektowaniu sieci, algorytmach uczenia i innych parametrach) stała się cenną umiejętnością, której głównym składnikiem jest w większej mierze szczególnego rodzaju intuicja niż twarda wiedza.

14



Przypadkowe jednostki

Modelowanie społeczeństwa

Modelowanie społeczeństwa jest dużo trudniejsze od modelowania procesów astronomicznych, fizycznych, chemicznych czy nawet biologicznych, ponieważ mamy tu do czynienia z nieprzewidywalnością ludzkich zachowań. Nieprzewidywalność ta bierze się stąd, iż człowiek jest istotą ogromnie złożoną, reagującą na bardzo wiele bodźców i reakcja ta zależy często od niedających się łatwo sklasyfikować czynników. Oczywiście, można i należy wykorzystać tu prawidłowości statystyczne, które pojawiają się w wyniku uśredniania po zbiorach zawierających dużo osobników. Całe szczęście, że społeczeństwo jest „przypadkowe", bo w przeciwnym razie przewidywanie byłoby jeszcze trudniejsze. To, co dla niektórych polityków jest nieszczęściem („Społeczeństwo składa się z przypadkowych jednostek, wciąż nie można mieć do niego zaufania" – to wyjątek z przemówienia sejmowego posłanki Haliny Nowiny-Konopczyny), staje się błogosławieństwem dla tych, którzy rozumieją działanie praw rządzących licznymi zbiorami obiektów. Właśnie dzięki owej przypadkowości można bowiem lepiej społeczeństwo opisać i zrozumieć, stosując do tego opisu prawa statystyczne. Przewidywania wysnute na tej podstawie z reguły jednak mają charakter krótkofalowy. Można przewidzieć wyniki wyborów, badając opinie wyborców na krótko przed wyborami, ale prognozowanie na kilka lat naprzód jest bardzo trudne. Przewidywanie zachowania społeczeństwa przypomina prognozowanie pogody. Trzepnięcie skrzydła motyla może zmienić trasę huraganu czy tajfunu i jedno zdanie wypowiedziane przez polityka może w dramatyczny sposób zmienić bieg historii. Ewolucja społeczeństwa oglądana z bliska ma niewątpliwie