Helion

KENNETH WENGER

CZY ALGORYTM SPISKUJE PRZECIWKO NAM?

CO KAŻDY POWINIEN WIEDZIEĆ
O KONCEPCJACH I PUŁAPKACH
SZTUCZNEJ INTELIGENCJI

Tytuł oryginału: Is the Algorithm Plotting Against Us?: A Layperson's Guide to the Concepts, Math, and Pitfalls of AI

Tłumaczenie: Grzegorz Werner

ISBN: 978-83-289-1636-4

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/czyalg
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Printed in Poland.

SPIS TREŚCI

1

POLARYZACJA I JEJ KONSEKWENCJE

Czym są sztuczne sieci neuronowe? Czy naprawdę są zbudowane z neuronów, tak jak nasze mózgi? Struktura sztucznych sieci neuronowych stanowi punkt wyjścia w dążeniu do ich zrozumienia, ale niewiele mówi o ich możliwościach i ograniczeniach. W tym celu musimy sięgnąć głębiej i zadać bardziej konkretne pytania. Dlaczego je wymyślono i jakie problemy można za ich pomocą rozwiązać? Skąd czerpią wiedzę? Czy mogą uczyć się same, czy trzeba je uczyć? W tym rozdziale poświęcimy sporo czasu na zrozumienie sztucznych sieci neuronowych i ich historii. Opowieść o ich początkach jest pełna optymizmu i nadziei.

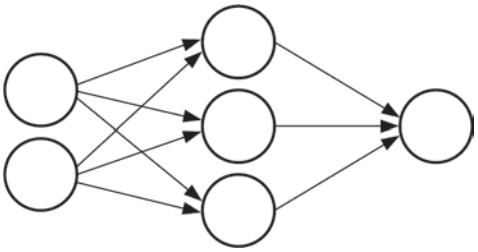
Jak się zapewne domyślasz, sztuczna sieć neuronowa, podobnie jak ludzki mózg, rozpoczęła swoje życie jako pojedyncza komórka. Z tej komórki rozwinęły się złożone sieci, które dziś są odpowiednikiem gwiazd rocka w świecie sztucznej inteligencji. Używamy ich do nadawania sensu danym. Wykorzystujemy je do klasyfikowania obrazów, rozpoznawania obiektów i prowadzenia pojazdów autonomicznych. Może widziałeś, jak tesla płynnie włącza się do ruchu bez interwencji człowieka. Samochód jest wyposażony w kamery i inne czujniki, które pomagają mu rejestrować stan świata zewnętrznego, ale zadaniem sztucznych sieci neuronowych jest interpretowanie danych z czujników w celu identyfikowania pojazdów, pieszych, znaków drogowych i wszystkiego, co pozwala im „widzieć" świat. Naukowcy pracują obecnie nad wykorzystaniem sieci neuronowych w medycynie jako klasyfikatorów obrazu ułatwiających diagnozowanie różnych rodzajów schorzeń, od czerniaka, przez raka piersi, po choroby układu oddechowego, takie jak zapalenie płuc. Na przykład zadaniem

sieci neuronowych może być identyfikowanie nietypowych obszarów na skanach tkanki pobranej podczas biopsji lub oznak wysięku płucnego na zdjęciach rentgenowskich. W bardziej ogólnym sensie używamy sztucznych sieci neuronowych do przetwarzania dużych ilości danych i wydobywania z danych wzorców i trendów, które mogą być dla nas istotne. W niektórych przypadkach wiąże się to z analizą obrazów, ale sieci neuronowe nie ograniczają się do zastosowań wizyjnych. Możemy wyszkolić sieć neuronową do przewidywania cen nieruchomości w określonych dzielnicach lub nauczyć ją analizowania danych historycznych z rynków finansowych w celu prognozowania przyszłych trendów. Innymi słowy, używamy sztucznych sieci neuronowych jako narzędzi do rozwiązywania pewnych klas problemów, a mianowicie *problemów klasyfikacji* i *problemów prognozowania*.

Jednak historia sztucznych sieci neuronowych w pewnym sensie nie jest historią poszukiwania sztucznej inteligencji. Wszystko zaczęło się od prób zrozumienia ludzkiego mózgu. Badacze lubią używać analogów i modeli do badania swoich koncepcji. Trudno jednak majstrować przy prawdziwym mózgu; ludzie są niewdzięcznym obiektem badań. Pionierzy neurologii i psychologii uznali więc, że przydałoby się zbudować sztuczny mózg, co byłoby krokiem w stronę zrozumienia rzeczywistego mózgu.

Aby odpowiedzieć na pytania z pierwszego akapitu, rozdział ten podzieliłem na kilka sekcji. Zaczniemy od spaceru po korytarzach historii i przyjrzymy się najwcześniejszym badaniom (z XIX wieku!), które przyczyniły się do zbudowania działającej sztucznej sieci neuronowej. Powiem, dlaczego badacze zaczęli postrzegać neuron jako kluczowy element w przetwarzaniu informacji, i przyjrzymy się pierwszemu sztucznemu neuronowi — neuronowi McCullocha-Pittsa. Dowiesz się, jak postępy w rozumieniu mózgu biologicznego przyczyniły się do modyfikacji neuronu McCullocha-Pittsa, która dała początek bardziej zaawansowanemu układowi zwanemu perceptronem, co doprowadziło do powstania najprostszego typu sztucznej sieci neuronowej, a mianowicie w pełni połączonej sieci neuronowej, znanej również jako perceptron wielowarstwowy (ang. *mutilayer perceptron*, MLP) (zob. rysunek 1.1). Wyjaśnię, dlaczego prawdopodobnie usłyszałeś o takich sieciach dopiero niedawno (choć badania nad sztucznymi sieciami neuronowymi rozpoczęły się na dobre w latach pięćdziesiątych XX wieku): badania w tej dziedzinie od samego początku miały wiele wzlotów i upadków, przy czym wiele z tych upadków wynikało z przesadzonych obietnic.

Spróbuję wykazać, że choć nadmierne oczekiwania wobec tej technologii nadal stanowią problem, to można się spodziewać, iż sztuczne sieci neuronowe zostaną z nami na dłużej. Po określeniu etapów, które doprowadziły do powstania nowoczesnych sztucznych neuronów, zbadamy, w jaki sposób neurony te można łączyć w złożone sieci. Wyjaśnię na przykładach, jak przetwarzane są informacje przez każdy neuron w sieci oraz jak oblicza się i interpretuje dane wyjściowe sieci. Rozdział zakończymy omówieniem kilku przykładowych zastosowań, które pokazują, jak można wykorzystać sieci neuronowe do rozwiązywania rzeczywistych problemów.



Rysunek 1.1. Prosta sztuczna sieć neuronowa. Informacje przepływają od lewej do prawej strony. *Dwa okręgi* w pierwszej warstwie to węzły wejściowe, *trzy okręgi* w środku to neurony przetwarzające, a *jeden okrąg* po prawej stronie reprezentuje wartość wyjściową. Wyjście w sieci neuronowe reprezentuje „prognozę" dla określonego wejścia

ROZPLĄTYWANIE NEURONU

Zanim przejdziemy do technicznych aspektów sieci neuronowych i ich funkcjonowania, poświęćmy kilka minut na zapoznanie się z ich historią. Zrozumienie, co robili pierwsi badacze, aby odcyfrować tajemnice mózgu, rzuci światło na to, dlaczego sieci neuronowe działają tak, a nie inaczej.

Aby zrozumieć sieci neuronowe, biologiczne czy sztuczne, musimy zacząć od neuronu. Obecnie wiemy, że nasze mózgi składają się z miliardów neuronów i że neurony są podstawowymi cegiełkami odpowiedzialnymi za przetwarzanie informacji. Ale nie zawsze o tym wiedzieliśmy. I jak to zwykle bywa w nauce, odkrycie neuronu i interpretacja jego funkcji nie pozostały bez dyskusji. Jak doszliśmy do uznania skromnego neuronu za podstawową jednostkę przetwarzania? Historia ta zaczyna się w XIX wieku od młodego, pełnego energii i entuzjazmu Hiszpana. Nazywał się Santiago Ramón y Cajal i jako pierwszy zrozumiał,

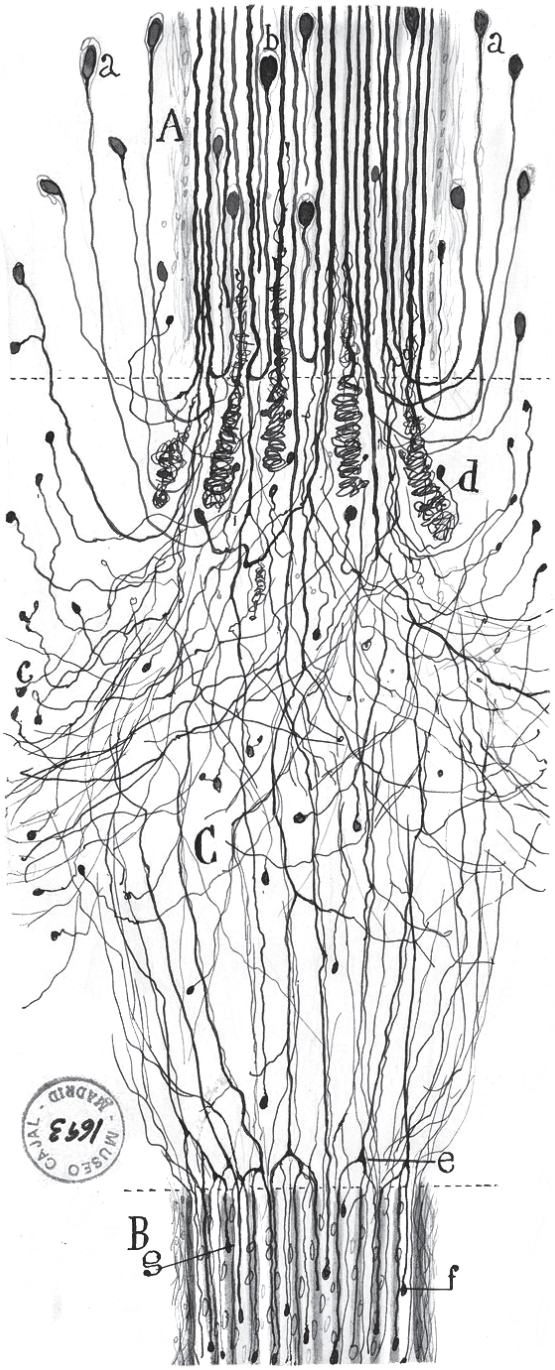
że mózg składa się z pojedynczych neuronów i że neurony te odgrywają kluczową rolę w przetwarzaniu informacji.

Cajal urodził się w 1852 roku w małej wiosce w Aragonii, w północno-wschodniej Hiszpanii. W młodości początkowo pragnął zostać artystą. Lubił rysować świat przyrody, co miało okazać się użyteczne w jego przyszłej karierze naukowej (zob. rysunki 1.2 – 1.5). Jego ojciec był chirurgiem i profesorem wydziału sekcyjnego na Uniwersytecie w Saragossie. Młodszy Cajal ostatecznie poszedł w ślady ojca i zapisał się do szkoły medycznej w Saragossie, którą ukończył w 1873 roku. Wielki wkład Cajala w neurobiologię rozpoczął się w 1887 roku. Wówczas Cajal udał się z Walencji do Madrytu, aby poznać nowe osiągnięcia technologiczne związane z przygotowaniem próbek do badania pod mikroskopem. Tam poznał genialnego psychiatrę Luisa Simarro Lacabrę, który pokazał mu próbki mózgu barwione techniką opracowaną 14 lat wcześniej przez Włocha Camillo Golgiego. Polegała ona na utwardzeniu kawałka materii mózgowej w roztworze dwuchromianu potasu, a następnie zanurzeniu go w roztworze azotanu srebra. Powodowało to wybarwienie tylko niektórych typów komórek i ujawniało ich kompletne struktury w postaci czarnych plam na niezabarwionym tle. Ci, którzy znali tę technikę — nieliczni, ponieważ przez kilkanaście lat od powstania nie cieszyła się ona dużą popularnością — nazywali ją *reazione nera* (czarną reakcją). Widząc okazy wypreparowane przez Lacabrę, Cajal szybko zdał sobie sprawę z niedoskonałości ówczesnych metod badania tkanki nerwowej. Jak później napisał w swojej autobiografii, technika barwienia spowodowała, że komórki „zabarwiły się na brunatno aż do najdrobniejszych gałązek, które wyróżniały się niezrównaną klarownością na przezroczystym żółtym tle. Wszystko było ostre jak szkic wykonany chińskim tuszem”[1]. Cajal wykorzystał i udoskonalił technikę barwienia, badając skład tkanki nerwowej siatkówki, móżdżku i rdzenia kręgowego.

[1] Autobiografia Cajala, *Recollections of My Life* (oryg. *Recuerdos de mi vida*), przetłumaczona na język angielski przez E.H. Craigiego we współpracy z J. Cano (MIT Press, Cambridge, MA, 1989); cytat pochodzi z artykułu M. Bentivoglio, *Life and Discoveries of Santiago Ramón y Cajal*, NobelPrize.org, 20 kwietnia 1998 [dostęp: 22 lipca 2024], *https://www.nobelprize.org/prizes/medicine/1906/cajal/article/*.

Rysunek 1.2. Komórki nowotworowe pokrywające błony mózgu, 1890. Instituto Cajal (CSIC), Madryt

Rysunek 1.3. Neuron Purkiniego z ludzkiego móżdżku, ok. 1900. Instituto Cajal (CSIC), Madryt



Rysunek 1.4. Przecięty nerw poza rdzeniem kręgowym, 1913. Instituto Cajal (CSIC), Madryt

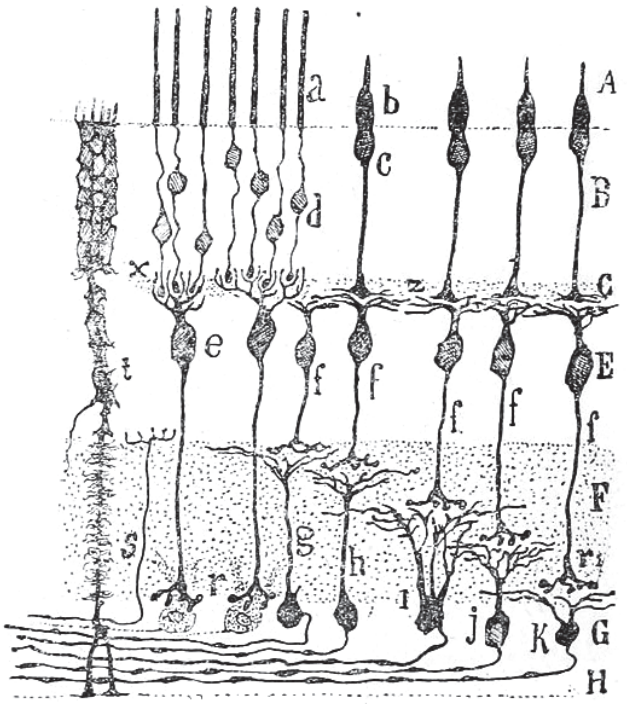


Fig. 27.
Coupe transversale de la rétine d'un mammifère.

Rysunek 1.5. „Coupe transversal de la retine d'un mammifere" (przekrój poprzeczny siatkówki oka ssaka). Ilustracja z książki *Les nouvelles idees sur la structure du systeme nerveux: chez l'homme et chez les vertebres*, C. Reinwald, Paryż 1894, s. 112

W tamtym czasie uważano, głównie za sprawą prac niemieckiego histologa Josepha von Gerlacha, że tkanka nerwowa składa się z komórek mających różne splątane wypustki, które tworzą ciągłą sieć zwaną *retikulum* (siateczką). Zgodnie z tym poglądem, w odróżnieniu od innych narządów ciała, które dzielą się na odrębne elementy, mózgu i układu nerwowego nie można rozdzielić na podstawowe i osobne jednostki. Camillo Golgi przeanalizował tkankę nerwową z użyciem własnych technik barwienia i zauważył, że komórki tkanki nerwowej mają dwa rodzaje wypustek: wiązkę krótkich włókien, które odchodzą od komórki i rozgałęziają się w wielu kierunkach, oraz jeden długi przewód, który ma nieliczne rozgałęzienia. Zauważył, że ciała poszczególnych komórek, chociaż rozgałęziają się w pobliżu innych podobnych komórek, w rzeczywistości nie

zlewają się ze sobą w ciągłe retikulum, ale opis Gerlacha był dla niego na tyle przekonujący, że wmówił sobie, iż długie połączenia wyrastające z komórek prawdopodobnie tworzą ciągłą ścieżkę w jakimś dalszym punkcie, którego nie dostrzegał.

To Cajal jako pierwszy uświadomił sobie indywidualność komórek układu nerwowego i, co ważniejsze, potencjalne konsekwencje takiej organizacji strukturalnej. Układ nerwowy opisywany jako retikulum był monolityczną reprezentacją niepodatną na ingerencję z zewnątrz, co groziło tym, że jego tajemnice pozostaną na zawsze ukryte w plątaninie tkanek. Pomysł, że układ nerwowy składa się z miliardów pojedynczych komórek, pomaga rozłożyć problem jego funkcjonowania na podstawowe elementy składowe i stanowi zrąb dzisiejszego rozumienia organizacji układu nerwowego.

W latach 1894 – 1904 Cajal stworzył jedno ze swoich najważniejszych dzieł, *Textura del Sistema Nervioso del Hombre y los Vertebrados* (Struktura układu nerwowego człowieka i kręgowców). Praca ta zawierała szczegółowe analizy i ilustracje struktur komórek nerwowych, możliwe dzięki artystycznym zamiłowaniom i młodzieńczym przygotowaniom Cajala. Jego rysunki do dziś są reprodukowane w podręcznikach neurologii. Jednym z najważniejszych osiągnięć Cajala jest *prawo dynamicznej polaryzacji*. Prawo to mówi, że komórki nerwowe ulegają polaryzacji poprzez rozdzielenie swojej głównej funkcji na dwie odrębne części ciała, wejściową i wyjściową. Komórki przyjmują sygnały wejściowe przez ciało i dendryty, a generują sygnały wyjściowe w aksonach.

Uznanie skromnego neuronu za dyskretny system zdolny do odbierania sygnałów wejściowych i wytwarzania sygnałów wyjściowych stało się podstawą zasady propagacji informacji w mózgu. Za swój wkład w neurobiologię Golgi i Cajal otrzymali w 1906 roku Nagrodę Nobla w dziedzinie fizjologii i medycyny. Wykład noblowski Golgiego obejmował opis neuronów tworzących retikulum. Co ciekawe, twierdzeniu temu całkowicie zaprzeczył wykład noblowski Cajala, który naturalnie skupiał się na roli neuronu jako odrębnej jednostki układu nerwowego, dalekiej od koncepcji jednego nieprzystępnego retikulum. Cajal walczył o swoje odkrycia i rozpowszechniał swoje idee aż do śmierci, która nastąpiła w 1934 roku. Idee te stanowią podstawę współczesnego rozumienia przetwarzania informacji w tkance nerwowej, a prace Cajala mają kluczowe znaczenie dla sztucznych sieci neuronowych.

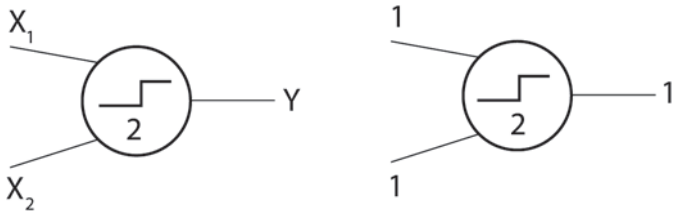
Trudno sobie wyobrazić wszystkie konsekwencje funkcji takiej prostej jednostki, która przyjmuje sygnał wejściowy i wytwarza sygnał wyjściowy. Trudno pojąć, że w sieci złożonej z miliardów takich połączeń może pojawić się inteligencja,

a nawet świadome zachowania. Naukowcy próbujący wyjaśnić, jak aktywność mózgu przekłada się na widoczne zachowania ludzi, zastanawiają się nad tym problemem od stuleci. Aby zrozumieć, jak działają złożone systemy, trzeba do nich zajrzeć, trzeba je zmierzyć, modyfikować ich organizację w sposób, który pozwala określić korelację między wyjściowym zachowaniem a stanem systemu w danym punkcie czasu. Oczywiście, jesteśmy ograniczeni, jeśli chodzi o takie badanie i analizowanie biologicznych mózgów (zwłaszcza ludzkich), ponieważ zmiany w tych obszarach zwykle prowadzą do poważnych skutków ubocznych. Potrzeba zrozumienia biologicznych procesów nerwowych doprowadziła do powstania narzędzi, które z czasem zapoczątkowały nową erę sztucznej inteligencji. Wszystko zaczęło się jednak od chęci zrozumienia mózgu. Świętym Graalem było zbudowanie modelu mózgu, przy którym można by majstrować.

SYGNAŁY I BRAMKI

W 1943 roku Warren McCulloch i Walter Pitts opublikowali artykuł badawczy zatytułowany *A Logical Calculus of Ideas Immanent in Nervous Activity*, w którym zaproponowali pierwszy model sztucznego neuronu. Był to bardzo prosty system: węzeł z wejściami, które mogły mieć wartość 0 lub 1, oraz progiem aktywacji, który mógł mieć dowolną wartość i służył jako mechanizm określający, kiedy neuron zostanie aktywowany. Aby nastąpiła aktywacja, suma wartości wejściowych musiała być co najmniej równa progowi.

Rozważmy na przykład neuron z dwoma wejściami X_1 i X_2, wartością progową równą 2 i wyjściem Y. Jeśli oba wejścia X_1 i X_2 są równe 1, to 1 + 1 = 2, co jest równe wartości progowej 2, neuron zatem aktywuje się, a Y = 1 (zob. rysunek 1.6). Jeśli oba wejścia są równe 0, to 0 + 0 = 0, co nie jest równe lub większe od wartości progowej 2, neuron zatem nie aktywuje się, a Y = 0. Co więcej, jeśli dowolny sygnał wejściowy jest równy 0, to 1 + 0 = 1, czyli mniej od progu aktywacji równego 2. Również w tym przypadku neuron nie wytwarza sygnału, a Y jest równe 0. Był to pierwszy działający sztuczny neuron, a kiedy następnym razem zobaczysz, jak Twoja roomba bodzie nogę stołu w pozornej sztucznej frustracji, wróć myślami do Santiago Ramóna y Cajala oraz neuronu McCullocha-Pittsa — bez nich roomba by nie istniała.



Rysunek 1.6. Neuron McCullocha-Pittsa z dwoma wejściami, progiem aktywacji 2 i jednym wyjściem. W przykładzie po prawej stronie oba sygnały wejściowe mają wartość 1

Neuron McCullocha-Pittsa to prosty system, który ma modelować najbardziej elementarne funkcje biologicznych neuronów. W 1943 roku, dzięki wcześniejszym pracom Cajala i sir Charlesa Scotta Sherringtona (laureata Nagrody Nobla za 1932 rok oraz autora książki *The Integrative Action of the Nervous System*), było już wiadomo, że biologiczne neurony odbierają sygnały wejściowe przez dendryty i że na podstawie pewnego wewnętrznego progu lub bardziej zniuansowanych obliczeń każdy neuron albo wysyła sygnał przez akson, albo nie. Jak widać, neuron McCullocha-Pittsa przybliża podstawowe elementy tego procesu. Przyjmuje zbiór wartości przez połączenia wejściowe i na podstawie prostego wewnętrznego stanu (wartości progowej) generuje na wyjściu 1 lub 0. Pytanie brzmi, jak użyteczny jest taki system. Czy naprawdę można zrobić coś interesującego z czegoś tak prostego? Okazuje się, że można.

Jedną z najbardziej użytecznych rzeczy, jaką można zrobić, jest zaimplementowanie zbioru funkcji logicznych zwanych *bramkami logicznymi*. Co interesujące, wszystkie współczesne urządzenia elektroniczne i komputerowe są oparte na bramkach logicznych. Bramki pomagają przekształcić płynące w przewodach zero-jedynkowe sygnały w stany, na podstawie których można podejmować decyzje. Aby zrozumieć, jak proste systemy, takie jak neurony, mogą wykonywać skomplikowane zadania, kiedy zostaną zintegrowane w złożoną sieć, dobrze jest wiedzieć, jak działają bramki logiczne. Najprostsze bramki logiczne to `NOT`, `AND` i `OR`.

Bramki AND

Bramka logiczna `AND` ma dwa przewody wejściowe (zob. rysunek 1.7). Kiedy oba przewody wejściowe są „włączone” (tzn. na obu jest sygnał reprezentujący wartość 1), wyjściem bramki `AND` jest 1. Kiedy oba przewody wejściowe są „wyłączone” (tzn. na obu jest sygnał reprezentujący wartość 0), a nawet kiedy tylko

Rysunek 1.7. Bramka logiczna AND z dwoma wejściami i jednym wyjściem. Jeśli którekolwiek z wejść ma wartość 0, wyjście ma wartość 0. Jeśli oba wejścia mają wartość 1, wyjście ma wartość 1

jeden z przewodów wejściowych jest „wyłączony", wyjściem bramki `AND` jest 0. Aby na wyjściu bramki `AND` pojawił się sygnał 1, zarówno wejście 1., jak i wejście 2. muszą być równe 1 (zob. tabela 1.1).

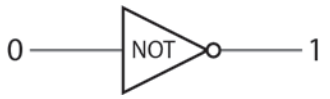
Tabela 1.1. Możliwe kombinacje wejścia i wyjścia bramki `AND`

Wejście 1	Wejście 2	Wyjście
0	0	0
0	1	0
1	0	0
1	1	1

Jak można to wykorzystać w praktyce? Przypuśćmy, że budujesz mechanizm bezpieczeństwa typowy dla maszyn przemysłowych — takich jak prasa hydrauliczna — który w celu zapewnienia, że ręce operatora będą znajdować się z dala od niebezpiecznego obszaru, ma dwa przyciski, które operator musi stale naciskać, aby maszyna działała. W takim systemie mógłbyś dodać bramkę logiczną `AND` z przewodami wejściowymi podłączonymi do dwóch przycisków na panelu sterowania i podłączyć wyjście bramki logicznej do sterownika maszyny. Tylko wtedy, gdy oba przyciski będą wciśnięte, oba wejścia bramki `AND` będą miały wartość 1, i tylko wtedy bramka wygeneruje na wyjściu wartość 1, sygnalizując, że maszyna może zacząć pracę. Jeśli nie jest wciśnięty żaden przycisk albo wciśnięty jest tylko jeden, bramka `AND` wygeneruje na wyjściu wartość 0, a maszyna pozostanie wyłączona.

Bramki NOT

Bramka `NOT` jest prawdopodobnie najprostszą z bramek logicznych i działa jak inwerter sygnału (zob. rysunek 1.8). Ma jeden przewód wejściowy i jeden wyjściowy. Jeśli wejście ma wartość 0, wyjście ma wartość 1. Jeśli wejście ma wartość 1, wyjście ma wartość 0 (zob. tabela 1.2).



Rysunek 1.8. Bramka logiczna NOT z jednym wejściem i jednym wyjściem. Jeśli wejście ma wartość 0, wyjście ma wartość 1. Jeśli wejście ma wartość 1, wyjście ma wartość 0

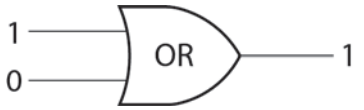
Tabela 1.2. Możliwe kombinacje wejścia i wyjścia bramki NOT

Wejście 1	Wyjście
0	1
1	0

Można to wykorzystać w wielu różnych sytuacjach. Przypuśćmy, że chcemy zaprojektować czujnik, który informuje, czy ilość paliwa w baku samochodu spadła poniżej pewnego poziomu. Możemy podłączyć czujnik paliwa do bramki NOT. Kiedy czujnik jest zanurzony w paliwie, wejście bramki NOT ma wartość 1, a wyjście ma wartość 0. Wyjściowy sygnał 0 jest interpretowany przez obwód sterujący jako wystarczający poziom paliwa. Kiedy tylko czujnik wynurzy się z paliwa, sygnał wejściowy bramki NOT przyjmie wartość 0, a wyjście zmieni się na 1. Bramka NOT wyśle do obwodu sterującego sygnał wskazujący, że kończy się paliwo.

Bramki OR

Bramka logiczna OR jest podobna do bramki AND (zob. rysunek 1.9). Ma dwa przewody wejściowe i jeden przewód wyjściowy. Jej wyjście ma wartość 1, kiedy „włączony” jest dowolny przewód wejściowy (zob. tabela 1.3).



Rysunek 1.9. Bramka logiczna OR z dwoma wejściami i jednym wyjściem. Jeśli dowolne z wejść ma wartość 1, wyjście ma wartość 1. Jeśli oba wejścia mają wartość 0, wyjście ma wartość 0

Tabela 1.3. Możliwe kombinacje wejścia i wyjścia bramki OR

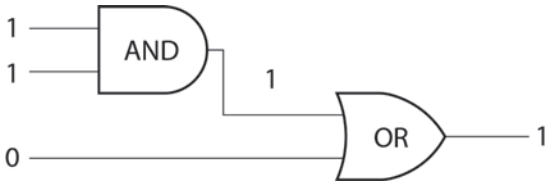
Wejście 1	Wejście 2	Wyjście
0	0	0
0	1	1
1	0	1
1	1	1

Możemy wykorzystać tę funkcję, aby w momencie zdarzenia podejmować działanie oparte na sekwencji przeszłych zdarzeń. Powiedzmy, że zlecono nam zaimplementowanie systemu płatności dla publicznej usługi transportowej, na przykład metra, a klienci mogą płacić za przejazd na dwa sposoby: przeskanować kartę płatniczą i przekazać środki elektronicznie albo zapłacić gotówką. System płatności można podłączyć do bramki OR tak, że system elektroniczny i system gotówkowy są podłączone do przewodów wejściowych bramki OR, a wyjście bramki jest podłączone do sterownika kołowrotu. Dopóki klient nie zapłaci albo elektronicznie, albo gotówką, bramka OR nie generuje sygnału wyjściowego (jej wyjście jest równe 0), a kołowrót pozostaje zablokowany. Kiedy tylko klient zapłaci albo elektronicznie, albo gotówką, sterownik kołowrotu otrzymuje sygnał (wyjście bramki przyjmuje wartość 1) i przepuszcza klienta.

Systemy bramek logicznych

Zapewne zaczynasz już rozumieć, jak duże możliwości mają pojedyncze bramki logiczne. I tak jak pojedyncze neurony łączą się w zaawansowane sieci, można również łączyć bramki logiczne, czyli używać wyjścia jednej bramki jako wejścia drugiej.

Rozważmy zautomatyzowany proces przyjmowania podróżnych na lotnisku w kraju należącym do Unii Europejskiej. (Przykład jest mało realistyczny, ale proszę o wyrozumiałość). Do kraju można dostać się na dwa sposoby: jeśli jesteś obywatelem UE, możesz od razu przejść przez stanowisko kontrolne, a jeśli nie jesteś obywatelem UE, musisz okazać ważną wizę i paszport. Moglibyśmy zbudować system przepuszczania podróżnych w oparciu o bramkę AND połączoną z bramką OR (zob. rysunek 1.10). Bramka AND jest podłączona do jednego z wejść bramki OR. Bramka AND generuje na wyjściu wartość 1, jeśli posiadasz paszport *oraz* ważną wizę UE. Drugie wejście bramki OR jest ustawiane na 1, jeśli jesteś



Rysunek 1.10. Bramka AND i bramka OR automatycznie kontrolujące wejście na terytorium UE

obywatelem UE. Zgodnie z tą logiką, jeśli jesteś obywatelem UE, co najmniej jedno z wejść bramki OR jest ustawione na 1, dlatego wyjście bramki OR jest równe 1 i możesz przejść. Jeśli nie jesteś obywatelem UE, jedno z wejść bramki OR ma wartość 0, zatem aby bramka OR wygenerowała na wyjściu wartość 1, jej drugie wejście musi mieć wartość 1. Drugie wejście zależy od wyjścia bramki AND, które ma wartość 1 tylko pod warunkiem, że posiadasz zarówno paszport, jak i ważną wizę UE.

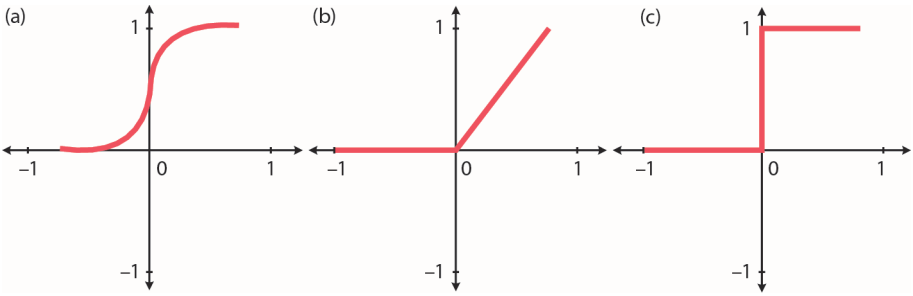
Możemy kontynuować łączenie bramek logicznych i budować w ten sposób złożone systemy. W rzeczywistości, jeśli połączymy wystarczającą liczbę bramek logicznych (miliony), zbudujemy nowoczesne komputery. Bramki logiczne to podstawowe elementy składowe systemów sterujących i decyzyjnych, które stanowią fundament języków programowania. Oprogramowanie składa się z szeregu instrukcji zrozumiałych dla komputera. Instrukcje te są czasami oparte na pewnych warunkach logicznych (np. „jeśli to, to tamto"). Badanie tych warunków jest możliwe dzięki bramkom logicznym. Kiedy czytamy o nowych procesorach komputerowych — CPU lub GPU — wyposażonych w miliony tranzystorów, zasadniczo właśnie to oznaczają te liczby. Tranzystory w procesorach służą głównie do budowania bramek logicznych, więc miliony tranzystorów odpowiadają milionom bramek logicznych. Ale po co w ogóle mówimy o bramkach logicznych?

Okazuje się, że bramki logiczne można budować z neuronów McCulloch-Pittsa, a koncepcja bramki logicznej była już dobrze znana w latach 50. ubiegłego wieku[2]. Naukowcy zauważyli więc, że skoro można budować bramki logiczne ze sztucznych neuronów, to neurony te można wykorzystać do skonstruowania innych złożonych systemów, może nawet modelu mózgu. Zmotywowało ich to do dalszych badań nad sztucznymi neuronami.

[2] Charles Babbage (powszechnie uważany za ojca komputera cyfrowego) użył wczesnych (mechanicznych!) bramek logicznych w latach 30. XIX wieku w swojej „maszynie analitycznej". W latach 20. i 30. XX wieku na użytek ówczesnych komputerów wynaleziono nowocześniejsze bramki logiczne.

Dostrajanie neuronu

Ograniczeniem neuronu McCullocha-Pittsa jest to, że przyjmuje on binarne dane wejściowe i generuje binarne dane wyjściowe. Nawet w latach 40. było wiadomo, że wejścia biologicznych neuronów nie zawsze mają charakter binarny (tzn. sygnał o pełnej intensywności albo sygnał o zerowej intensywności). W biologicznych neuronach sygnały są zniuansowane i mają pewien zakres intensywności. Co więcej, w neuronie McCullocha-Pittsa decyzja o tym, czy neuron się aktywuje, czy nie — nazywana też *funkcją aktywacji* neuronu — jest funkcją progową, co również zbytnio nie przypomina biologicznych neuronów (zob. rysunek 1.11).



Rysunek 1.11. Trzy funkcje aktywacji omawiane w tym rozdziale. Sigmoid (a) generuje na wyjściu wartość z zakresu od 0 do 1 dla wszystkich wyjść, przy czym wejście 0 daje na wyjściu wartość 0,5. Prostownik liniowy (b) daje na wyjściu wartość 0 dla wszystkich wejść ujemnych oraz niezmodyfikowaną wartość wejściową dla wszystkich wejść dodatnich. Funkcja progowa (c) daje na wyjściu 0 dla wszystkich wartości wejściowych mniejszych od wartości progowej, a 1 dla wszystkich wartości wejściowych większych od wartości progowej

W 1949 roku Donald Hebb, kanadyjski psycholog badający udział neuronów w procesie uczenia się, napisał książkę zatytułowaną *The Organization of Behaviour* (*Organizacja zachowań*). Przedstawił w niej teorię uczenia się, która zrewolucjonizowała poglądy na przetwarzanie informacji w tkankach nerwowych i pomogła zmienić sposób implementowania sztucznych neuronów. W swojej książce Hebb pisze: „Kiedy akson komórki A znajduje się wystarczająco blisko, aby pobudzić komórkę B, i wielokrotnie lub stale bierze udział w jej pobudzaniu, w jednej lub obu komórkach zachodzi pewien proces wzrostu lub zmiany metaboliczne, tak że skuteczność A, jako jednej z komórek aktywujących B, wzrasta”[3].

[3] D.O. Hebb, *The Organization of Behaviour: A Neuropsychological Theory*, John Wiley & Sons, Nowy Jork, Chapman & Hall, Londyn 1949, s. 62.

Hebb twierdzi, że wspólna aktywacja neuronów nie tylko wzmacnia połączenie między neuronami, ale sama stanowi podstawowy krok w procesie uczenia się. Sugeruje to, że połączeniom z neuronami powinny towarzyszyć wagi, które modulują znaczenie każdego połączenia na przestrzeni czasu. Informacje przepływające przez sieć neuronową zależą zatem nie tylko od wartości wejściowych — połączenia między neuronami są wzmacniane lub osłabiane w procesie uczenia się poprzez dodanie wagi do każdego połączenia. Wagi wpływają na to, jaka część wartości wchodzącej do danego połączenia zostanie odebrana przez neuron. Dodanie wag oznacza, że sieci neuronowe nie są trwałe i niezmienne. Można je regulować, a informacje można kierować i przekierowywać poprzez wzmacnianie lub osłabianie połączeń między neuronami, podobnie jak rzeka zmienia swój bieg w zależności od napotkanych przeszkód.

Zastanów się przez chwilę nad tym odkryciem. Było ono niezmiernie ważne. Dodanie wag do połączeń przypominało dodanie pokrętła strojenia do radia. Radio zbudowane do odbioru określonej częstotliwości, bez pokrętła, jest jak neuron McCullocha-Pittsa. Trzeba by przebudować neurony, żeby zmienić ich wyjścia, podobnie jak trzeba by przebudować radio bez pokrętła, żeby zmienić jego częstotliwość. Dodanie wag do neuronu oznacza, że możemy przekręcać „pokrętła" i zmienić zachowanie neuronów w czasie wykonywania programu, podobnie jak możemy zmienić częstotliwość odbieraną przez radio.

PERCEPTRON

Frank Rosenblatt, amerykański psycholog pracujący w 1958 roku w Laboratorium Aeronautycznym Uniwersytetu Cornella w ramach projektu finansowanego przez Biuro Badań Marynarki Wojennej Stanów Zjednoczonych, przestudiował prace McCullocha i Pittsa związane ze sztucznym neuronem oraz odkrycia Hebba i opracował **perceptron**. Perceptron był modyfikacją neuronu McCullocha-Pittsa, która obejmowała ważone połączenia wejściowe Hebba, a także umożliwiała każdemu indywidualnemu wejściu przyjmowanie wartości od 0 do 1, zamiast podejścia binarnego (wszystko albo nic). Zmiany te sprawiają, że perceptron jest bardziej podobny do neuronów biologicznych.

Historia sztucznych sieci neuronowych jest pełna subtelnych, ale kluczowych odkryć. Dodanie wag i liczb rzeczywistych (liczb z częścią ułamkową, które pozwalają mierzyć wielkości ciągłe, np. wszystkie liczby od 0 do 1, a nie tylko 0 lub 1)

sugerowało, że możliwe będzie zbudowanie ogólnych modeli mózgu, które można by regulować w celu rozwiązywania problemów ogólnych, a nie tylko konkretnych zadań. Jak zauważył Rosenblatt w swojej książce *Principles of Neurodynamics* z 1961 roku, istnieją dwa rodzaje modeli przeznaczonych do badania mózgu: monotypowe i genotypowe.

Modele monotypowe przypominają wstępnie dostrojone radio bez pokręteł, natomiast modele genotypowe są adaptowalne i można dostrajać je do różnych „częstotliwości" bez budowania ich od nowa. W modelu monotypowym zaczynamy od wymagań funkcjonalnych jakiejś „funkcji psychologicznej"[4], na przykład dobrze zdefiniowanego algorytmu rozpoznawania z określonymi warunkami wejściowym i wyjściowymi. Oznacza to, że w modelu monotypowym znajdujemy funkcję psychologiczną, którą chcemy wymodelować. Przypuśćmy, że chcemy modelować przetwarzanie zapachów w mózgu. Moglibyśmy opracować sztuczny system zdolny do wykrywania pewnych typów zapachów, a następnie zaprojektować konkretną reakcję na każdy zapach. Po uruchomieniu naszego sztucznego systemu monitorujemy proces wykrywania zapachów i reakcje na każdy zapach u ludzkiego ochotnika i próbujemy znaleźć podobieństwa między systemem sztucznym a biologicznym. Monotypowe podejście do studiowania mózgu zaczyna się od określonego zbioru wejść i określonego zbioru pożądanych wyjść, a po zbudowaniu systemu, który odwzorowuje te wejścia na pożądane wejścia, próbuje znaleźć podobne zachowania w systemach biologicznych. Następnie możemy zaktualizować model zgodnie z tym, co widzimy w systemach biologicznych. Podejście to lepiej pasuje do neuronu McCullocha-Pittsa. Jest to przemyślana metoda, w której dokładnie wiemy, jakiego rodzaju rozwiązanie budujemy.

W podejściu genotypowym, zamiast zaczynać od dobrze zdefiniowanych funkcji i porównywać nasze sztuczne systemy z mózgiem, zaczynamy od zbioru ogólnych reguł uczenia się i budujemy bardziej ogólny algorytm, który może nauczyć się modelowania dowolnych problemów z użyciem uniwersalnej procedury treningowej (a nie procedury specyficznej dla danego problemu).

[4] F. Rosenblatt, *Principles of Neurodynamics*, Cornell Aeronautical Laboratory, Buffalo, NY 1961, s. 11. Wszystkie pozostałe cytaty w tym podrozdziale pochodzą z artykułu M. Lefkowitz, *Professor's Perceptron Paved the Way for AI — 60 Years Too Soon*, Cornell Chronicle, 25 września 2019 [dostęp: 22.07.2024], *https://news.cornell.edu/stories/2019/09/professors-perceptron-paved-way-ai-60-years-too-soon*.

Na przykład zamiast budować system do wykrywania określonego zakresu zapachów, inny system do wykrywania zdefiniowanego zbioru smaków i jeszcze inny do wykrywania pożądanego zakresu dźwięków, podejście genotypowe próbuje znaleźć ogólny system do wykrywania bodźców wejściowych, którymi mogą być zapachy, smaki lub dźwięki. Podejście genotypowe nadaje większą plastyczność strukturze sztucznych sieci neuronowych, ponieważ projekt nie musi blisko odzwierciedlać zbioru początkowych wymagań — zamiast tego sieć może uczyć się, a przez to sama określić wymagania, które pozwalają jej osiągnąć wyjściowe cele. Dzięki zastosowaniu liczb rzeczywistych (z zakresu od 0 do 1) w połączeniach sieciowych zamiast wartości logicznych (0 lub 1, prawda lub fałsz) używanych przez neuron McCullocha-Pittsa, a także dodaniu wag do połączeń synaptycznych, perceptron Rosenblatta nadaje się lepiej do implementowania modeli genotypowych.

Rosenblatt uważał podejmowanie decyzji i inteligencję w mózgu za zgodne z zestawem algorytmów statystycznych i probabilistycznych, w których zamiast odwzorowywać zestaw bodźców wejściowych na określony zestaw wyjściowych zachowań psychologicznych (modele monotypowe), mózg przypisuje klasy danych wejściowych do klas wyników (modele genotypowe). Istnieje bardzo istotna różnica pomiędzy tymi dwoma podejściami. Gdyby mózg funkcjonował wyłącznie w oparciu o odwzorowywanie konkretnych danych wejściowych na konkretne wyjścia — zestaw predefiniowanych algorytmów niezbędnych do realizowania wszystkich funkcji, które wykonujemy codziennie i przez całe życie — nasze mózgi potrzebowałyby oszałamiająco dużego zbioru dyskretnych algorytmów. Oznaczałoby to niemal na pewno, że sztuczne systemy zdolne do naśladowania inteligencji na poziomie ludzkim pozostałyby w sferze marzeń. Gdyby jednak mózg funkcjonował jako system statystyczny odwzorowujący klasy danych wejściowych na klasy wyników, nie potrzebowalibyśmy konkretnego algorytmu dla każdej wykonywanej przez nas funkcji — potrzebowalibyśmy po prostu algorytmu dla każdej klasy wykonywanych funkcji. Pozwala to przynajmniej zmniejszyć liczbę systemów, które musimy emulować, jeśli chcemy zbudować sztuczną inteligencję na poziomie ludzkim.

Oczywiście mózg (i natura) nie ma obowiązku słuchać Rosenblatta i może być tak, że mózg stosuje oba podejścia — specyficzne algorytmy dla specyficznych funkcji oraz algorytmy ogólne dla większości pozostałych funkcji. Rosenblattowi chodziło o to, że jeśli chcemy zbudować system przypominający sztuczną inteligencję, a i tak nie wiemy, jak dokładnie działa mózg, to załóżmy, że działa

w sposób, który daje nam większą szansę emulowania go, i zobaczmy, dokąd nas to zaprowadzi. Trzeba zauważyć, że Rosenblatt nie zbudował perceptronu, przynajmniej początkowo, z myślą o stworzeniu sztucznej inteligencji. Jego podstawowym celem było zbudowanie systemu, przy którym można by było majstrować, żeby lepiej zrozumieć działanie mózgu.

Jak zatem działał ten perceptron? Perceptron Rosenblatta funkcjonował w następujący sposób: w perceptronie z pięcioma wejściami X_1, X_2, X_3, X_4, X_5 z każdym połączeniem wejściowym (analogicznym do synaps biologicznego neuronu) związana jest waga (w_1, w_2, …, w_5). Wagi również są liczbami rzeczywistymi, co oznacza, że mogą przyjmować dowolną wartość od 0 do 1 (zauważ, że wartości mogą być również większe od 1, ale zwykle normalizuje się je do zakresu 0 – 1). Proces obliczania wyjścia perceptronu polega na obliczeniu ważonej sumy wejść i zastosowaniu funkcji aktywacji do uzyskanego wyniku. Klasyczną funkcją aktywacji w perceptronie była funkcja progowa, tak jak w przypadku neuronu McCullocha-Pittsa, ale jak widać na rysunku 1.11, można zastąpić ją innymi funkcjami. W przypadku funkcji progowej sprawdzamy, czy ważona suma jest większa od wartości progowej. Jeśli jest, neuron generuje na wyjściu wartość 1, w przeciwnym razie generuje wartość 0. *Sumę ważoną* oblicza się według wzoru:

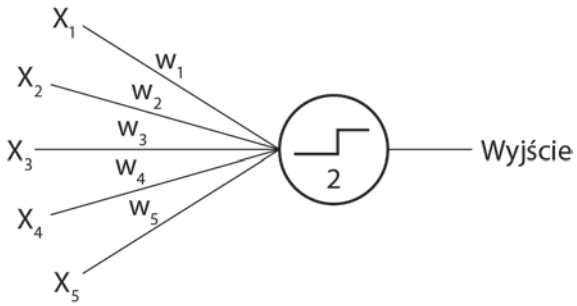
$$V = \left(\sum_{j=1}^{n} w_j x_j \right) + b$$

Formuła wygląda na skomplikowaną, ale pamiętaj, że Σ oznacza po prostu sumę, więc równanie mówi, iż wartość wyjściowa (V) jest sumą wartości wejściowych (x) pomnożonych przez wagi (w) połączeń dla wszystkich wejść (j) powiększoną o wartość b. O wartości b powiem więcej w rozdziale 3., na razie wystarczy wiedzieć, że jest nazywana *biasem* (przesunięciem) i, jak widać, nie wnosi dużo do wyniku w porównaniu z połączeniami wejściowymi i wagami[5].

[5] Bias jest bez wątpienia potrzebnym wyrazem i w zależności od skali danych może nabrać dużego znaczenia. Jednak chwilowo go zignorujemy ze względu na złożone obliczenia wag połączeń. Matematyka i tak jest dość skomplikowana, a uwzględnienie biasu w naszych obliczeniach mogłoby utrudnić zrozumienie tekstu początkującym Czytelnikom. Pamiętaj jednak, że zgodnie z równaniem należałoby dodać wyraz biasu po każdym zbiorze transformacji w danej warstwie sieci.

W następnych dwóch rozdziałach wyjaśnię, jak działają sieci neuronowe na poziomie pojedynczych neuronów, więc skupimy się na najważniejszej części procesu wejścia-wyjścia, tzn. na wejściach, wagach i funkcjach aktywacji. Po obliczeniu ważonej sumy wejść bierzemy wynikową wartość V i podstawiamy ją do funkcji aktywacji $O = f(V)$, która daje nam wartość wyjściową (O) neuronu.

Aby obliczyć wyjście perceptronu pokazanego na rysunku 1.12, wykonujemy następujące działanie: $O = f(w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5) + b$, gdzie f jest progową funkcją aktywacji z wartością progową równą 2. Neuron generuje na wyjściu wartość 1, jeśli wynik sumowania jest większy lub równy 2, a w przeciwnym razie generuje wartość 0. Wszystko to brzmi świetnie, ale skąd biorą się te magiczne wartości wag? Początkowe wagi perceptronu wybiera się losowo, a następnie reguluje się je w fazie trenowania modelu, porównując wyjście z oczekiwanymi wynikami i aktualizując wagi tak, aby model działał coraz lepiej z każdą iteracją cyklu treningowego. Tym jednak zajmiemy się później.



Rysunek 1.12. Perceptron Rosenblatta z pięcioma połączeniami wejściowymi. Z każdym połączeniem związana jest waga (w). Wartość wagi może być dowolną liczbą z zakresu od 0 do 1. Wyjście perceptronu zależy od tego, czy ważona suma połączeń przekroczy określony próg

Jeśli zastanawiasz się, czy ten perceptron ma zastosowania praktyczne, czy raczej jest po prostu ciekawostką naukową, to wiedz, że z pewnością przydaje się do rozwiązywania pewnych typów problemów. Perceptron po raz pierwszy zaimplementowano jako oprogramowanie do komputera IBM 704. Było to urządzenie wielkości pokoju, które wykorzystywało karty perforowane jako interfejs użytkownika. Perceptron ten wytrenowano tak, aby odróżniał karty oznaczone po lewej stronie od kart oznaczonych po prawej stronie, co w tamtych czasach było niewiarygodnym osiągnięciem. Rosenblatt nazwał perceptron „pierwszą maszyną, która może wpaść na oryginalny pomysł". Pierwszą implementacją

perceptronu jako narzędzia do automatyzacji był Mark 1, opracowany w 1958 roku — w tym przypadku była to maszyna, a nie oprogramowanie. Był zaprojektowany do rozpoznawania obrazów i potrafił analizować obrazy wejściowe o szerokości 20 pikseli i wysokości 20 pikseli. Wejściem perceptronu była kamera z fotokomórkami zbudowanymi z siarczku wapnia. 400 (20×20) fotokomórek losowo podłączono do kanałów wejściowych perceptronu, tzn. system używał perceptronu mającego 400 wejść. Wagi połączeń z perceptronem były zakodowane w macierzy specjalnych rezystorów zwanych potencjometrami, które można było aktywnie regulować, aby zmieniać ich opór, a przez to modyfikować napięcie.

W systemach komputerowych wartości 0, 1 i dowolne pośrednie są reprezentowane przez napięcia. Potencjometry pozwalały regulować wagę połączenia w czasie działania systemu, a przez to reprezentować dowolną wartość z zakresu od 0 do 1. W fazie uczenia perceptronu potencjometry regulowano z użyciem siników elektrycznych tak, aby ustawić odpowiednią wartość napięcia dla każdej wagi. W 1959 roku Bernard Widrow i Marcian Hoff z Uniwersytetu Stanforda wykorzystali postępy w badaniach nad perceptronem do zbudowania pierwszych sieci neuronowych, które rozwiązywały rzeczywisty problem. Były to sieci neuronowe wytrenowane do eliminowania szumu z linii telefonicznych. Systemy te, które nazwano ADALINE i MADALINE (od ang. *ADAptive LINear Elements* oraz *Multiple ADAptive LINear Elements*), są w użyciu po dziś dzień!

Wydawało się, że sztuczne sieci neuronowe mają się doskonale, a prawdziwa sztuczna inteligencja — łącznie z antropoidalnymi androidami gotowymi do wykonywania naszych rozkazów — jest tuż za rogiem. Co więc poszło nie tak?

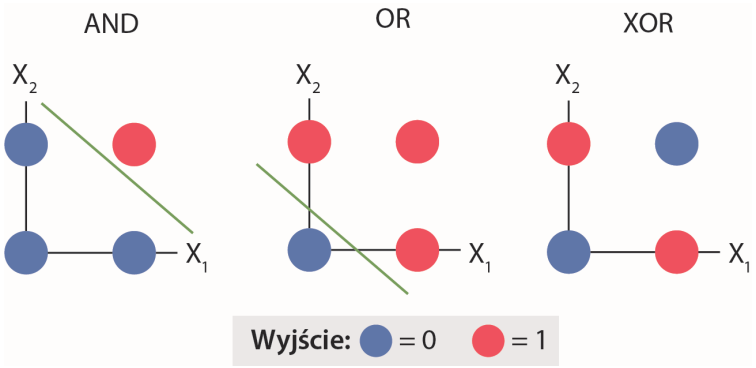
Mniej więcej w tym czasie entuzjazm związany z nową erą interakcji między człowiekiem a maszyną sięgnął zenitu. „New York Times" opublikował artykuł zatytułowany *Nowe urządzenie marynarki wojennej uczy się poprzez działanie: psycholog pokazuje zarodek komputera, który ma czytać i stawać się coraz mądrzejszy*, a w „New Yorkerze" napisano: „Rzeczywiście wygląda na to, że jest to pierwszy poważny rywal ludzkiego mózgu". Sam Rosenblatt nie próbował hamować przesadnych oczekiwań, wygłaszając stwierdzenia w rodzaju: „Niebawem będziemy świadkami narodzin takiej maszyny — maszyny zdolnej do postrzegania, rozpoznawania i identyfikowania otoczenia bez żadnego szkolenia i kontroli ze strony człowieka"[6].

[6] Lefkowitz, *Professor's Perceptron*.

CALL OF DUTY RATUJE SIEĆ NEURONOWĄ

Niestety dla Rosenblatta i dla sztucznych sieci neuronowych nadmierna uwaga, którą w krótkim czasie przyciągnęły te systemy, poirytowała badaczy sztucznej inteligencji stosujących bardziej tradycyjne podejścia. Najbardziej znanym spośród tych badaczy był Marvin Minsky, który w 1969 roku wspólnie z roku Seymourem Papertem wydał książkę pod tytułem *Perceptrons*. Minsky wziął w niej na cel perceptron i jego niezdolność do rozróżniania klas nierozdzielnych liniowo. System, który Rosenblatt zaprojektował do rozpoznawania obrazu z użyciem kamery o rozdzielczości 20×20 pikseli, używał perceptronu jednowarstwowego, a perceptrony jednowarstwowe są ograniczone do problemów rozdzielnych liniowo. Oznacza to, że jeśli wyobrazisz sobie klasyfikator jako linię oddzielającą dwie klasy obiektów (powiedzmy kropki niebieskie od czerwonych), linia może oddzielić te dwie klasy obiektów tylko pod warunkiem, że są one ułożone w sposób pozwalający na bisekcję zajmowanej przez nie przestrzeni. Weźmy grupę niebieskich i czerwonych kulek rozsypanych na blacie. Możemy umieścić patyk pomiędzy czerwonymi i niebieskimi kulkami tylko pod warunkiem, że kulki są już pogrupowane według kolorów. Jeśli kulki są wymieszane na powierzchni stołu, nie możemy położyć patyka w taki sposób, aby oddzielić kulki niebieskie od czerwonych. Właśnie za to Minsky skrytykował perceptron jednowarstwowy. Mówiąc ściślej: stwierdził, że perceptron jednowarstwowy nie może implementować bramki logicznej z funkcją alternatywy rozłącznej (`XOR`).

Bramki logiczne omówiłem wcześniej w tym rozdziale. W opisie bramki `OR` stwierdziliśmy, że bramka generuje na wyjściu wartość 1, jeśli albo wejście X_1, albo wejście X_2, albo oba te wejścia mają wartość 1. Bramka `XOR` generuje sygnał wyjściowy 1 tylko wtedy, kiedy albo wejście X_1, albo wejście X_2 ma wartość 1, ale nie wtedy, kiedy oba mają wartość 1 — na tym polega jej „rozłączność". Okazuje się, że perceptron jednowarstwowy nie może zaimplementować tej bramki logicznej. Aby zrozumieć dlaczego, pomyśl o perceptronie jako o klasyfikatorze liniowym. Jeśli narysujesz możliwe wyjścia bramki w zależności od możliwych wejść bramki, przekonasz się, że dla bramek `AND` i `OR` możesz łatwo narysować linię oddzielającą wyjścia 0 i 1, ale dla bramki `XOR` nie da się narysować jednej linii oddzielającej dwie klasy wyjść (zob. rysunek 1.13). Ma to daleko idące konsekwencje, ponieważ problemy, które musimy rozwiązywać w świecie rzeczywistym, najczęściej nie są rozdzielne liniowo.



Rysunek 1.13. Możliwe wyjścia bramek logicznych AND, OR i XOR. Bramka AND generuje wartość 1 tylko wtedy, gdy zarówno wejście X_1, jak i X_2 są równe 1 — pokazano to jako *czerwony okrąg*; w przeciwnym razie bramka generuje wartość 0 (*niebieski okrąg*). Wyjścia bramki AND są rozdzielne liniowo: można narysować jedną linię, która oddziela dwie klasy wyjść, 0 i 1. To samo dotyczy bramki OR. Ale w przypadku bramki XOR nie można narysować jednej linii, która oddzielałaby dwie klasy wyjść, wyjścia bramki XOR są zatem nierozdzielne liniowo

Owszem, Minsky zdawał sobie sprawę, że problem ten można rozwiązać przez połączenie większej liczby warstw perceptronu i utworzenie bardziej złożonej sieci neuronowej. Problem, jak zauważył Minsky, polegał na tym, że trening sieci neuronowej wystarczająco złożonej, aby rozwiązywać problemy nierozdzielne liniowo, takie jak klasyfikacja obrazów o wyższej rozdzielczości, byłby niewykonalny w praktyce, tzn. obliczenie prawidłowych wartości wag dla każdego połączenia między neuronami w każdej warstwie zajęłoby zbyt dużo czasu. Ta krytyka zapoczątkowała epokę znaną jako „zima AI". Przez następne 12 lat rządowe źródła finansowania badań nad sieciami neuronowymi niemal zupełnie wyschły. Na tym właśnie polega problem szumu medialnego, na który nauka bynajmniej nie jest odporna. Postęp naukowy zazwyczaj przebiega po gładkiej krzywej, a nowe odkrycia z wolna uzupełniają poprzednie — jest to proces stopniowy. Znacznie rzadziej zdarza się rozwój skokowy, kiedy to nowe odkrycie znacząco zmienia ugruntowaną wiedzę. Gdy niektórzy naukowcy i media wyolbrzymiają znaczenie nowych technologii lub odkryć, to przyczyniają się do ich upadku, kiedy rezultaty nie spełnią oczekiwań społeczeństwa. Łatwo bawić się w rewizjonizm historyczny, ale może atak Minsky'ego nie byłby tak zabójczy, gdybyśmy nie zachłysnęli się perceptronem jako ostatecznym rozwiązaniem sztucznej inteligencji, a po prostu uznali go za mały krok we właściwym kierunku.

Na szczęście po pewnym czasie nastąpił renesans sieci neuronowych. Nastroje zaczęły się zmieniać w 1982 roku, kiedy John Hopfield zaprezentował sieć Hopfield Net w Narodowej Akademii Nauk. Mniej więcej w tym czasie Japonia zapowiedziała projekt komputera piątej generacji, mający na celu wzmocnienie krajowych badań nad sztuczną inteligencją, w tym sieciami neuronowymi. Ogłoszono to podczas wspólnej amerykańsko-japońskiej konferencji poświęconej kooperatywno-konkurencyjnym sieciom neuronowym, co skłoniło Departament Obrony Stanów Zjednoczonych do wznowienia badań nad sieciami neuronowymi. Wkrótce ponownie zaczęły płynąć fundusze mające na celu utrzymanie przewagi technologicznej Stanów Zjednoczonych w obliczu postępów Japonii. Okresy zawyżonych oczekiwań i sowitego finansowania, po których następowało rozczarowanie i wyczerpanie funduszy, przeplatały się przez całe lata 90. i początek XXI wieku. W 2012 roku Alex Krizhevsky, Ilya Sutskever i Geoffrey E. Hinton z Uniwersytetu w Toronto napisali artykuł zatytułowany *ImageNet Classification with Deep Convolutional Neural Networks*, który pomógł ponownie rozbudzić zainteresowanie sztuczną inteligencją; utrzymuje się ono do dziś i wydaje się, że tym razem szybko nie przeminie. Sieci neuronowe rozwiązują już rzeczywiste problemy i chociaż ich przyszłość i potencjał są przedmiotem wielu spekulacji, oferowane przez nie rozwiązania będą paliwem dalszego zrównoważonego rozwoju. Sytuacja w mediach nie poprawiła się znacząco w ciągu ostatnich kilku dekad, ale zdarzyło się coś nieoczekiwanego, co sprawiło, że sieci badawcze są wciąż intensywnie studiowane prawie 10 lat po ich ponownym odkryciu.

Co zatem zmieniło się w 2012 roku? W latach 60. badacze wiedzieli, że połączenie szeregu wielowarstwowych perceptronów i utworzenie bardziej złożonej sieci znanej jako perceptron wielowarstwowy (ang. *multilayer perceptron*, MLP) pozwoliłoby rozwiązywać zagadnienia nieliniowe. Problemem, jak wskazał Minsky, było to, że proces trenowania sieci — innymi słowy, znajdowania właściwej wartości wagi każdego połączenia — zająłby setki lub tysiące lat czasu obliczeniowego, przez co był niepraktyczny. Jednak w latach 60. komputery były znacznie bardziej ograniczone. Ponadto badacze myśleli o obliczeniach wykonywanych przez standardowe procesory (ang. *central processor unit*, CPU). Procesor jest głównym komponentem przetwarzania danych w systemie komputerowym, a instrukcje są wykonywane w nich w sposób sekwencyjny. To znaczy, że jeśli chcemy wykonać dziesięć operacji, procesor zaczyna od pierwszej, potem

przechodzi do drugiej itd. Kiedy problem, który próbujemy rozwiązać, wymaga milionów obliczeń (jak w przypadku złożonych sieci neuronowych), a obliczenia te są wykonywane sekwencyjnie, to czas potrzebny do wytrenowania sieci byłby niepraktycznie długi nawet w przypadku szybkich współczesnych procesorów. Jednak w miarę jak moc procesorów rosła pod koniec lat 90. i na początku lat 2000., niektóre duże klastry procesorowe zwane *superkomputerami* stały się zdolne do wykonywania ogromnej ilości obliczeń, więc zainteresowanie sieciami neuronowymi znów zaczęło z wolna rosnąć. Ponieważ jednak nie każdy miał dostęp do superkomputera, sieci neuronowe wciąż stanowiły margines badań nad sztuczną inteligencją. Na szczęście postępy w zupełnie innej branży miały niebawem wydobyć sieć neuronową z zapomnienia i wciągnąć ją z powrotem do głównego nurtu.

Odkąd w latach 80. pojawiły się pierwsze rozpikselowane gry, branża gamingowa naciskała na producentów procesorów graficznych (ang. *graphics processing unit*, GPU), aby produkowali coraz szybszy i bardziej efektywny sprzęt. Interesującym aspektem grafiki komputerowej jest to, że operacje odbywają się na poziomie pojedynczych pikseli. Jeśli spojrzymy na grę komputerową z perspektywy czysto graficznej, zobaczymy serię klatek, które muszą zostać narysowane na ekranie. (Podobnie jak filmowy kadr klatka to obraz wypełniający ekran i przedstawiający ewoluującą scenę). Klatki składają się z tysięcy (lub milionów) pikseli. Zwiększenie wydajności i efektywności grafiki komputerowej wymagało stworzenia procesorów, które mogłyby zrównoleglić pracę związaną z rysowaniem tych pikseli. Oznaczało to, że w 2012 roku najpotężniejszymi dostępnymi procesorami poza superkomputerami nie były CPU, lecz GPU. Jeśli chodzi o pracę związaną z trenowaniem sieci neuronowej, możemy myśleć o CPU i GPU jak o potężnych kalkulatorach. Różnica między CPU a GPU polega na tym, że CPU przeprowadza wszystkie obliczenia sekwencyjnie, a GPU potrafi wykonywać tysiące operacji równolegle; GPU może zatem przetworzyć zbiór operacji znacznie szybciej niż CPU.

Krizhevsky, Sutskever i Hinton byli tego świadomi, kiedy postanowili wytrenować złożoną sieć neuronową z użyciem GPU. W 2012 roku ich implementacja sieci neuronowej o nazwie AlexNet wygrała konkurs ImageNet Large Scale Visual Recognition Challenge, uzyskując wyniki lepsze od wszystkich innych modeli do rozpoznawania obrazów. Warto zauważyć, że obok postępów w technologii GPU duże znaczenie dla sukcesu Krizhevsky'ego, Sutskevera i Hintona

miał też sam zbiór danych ImageNet. W 2006 roku Fei-Fei Li, badacz z Uniwersytetu Princeton, rozpoczął pracę nad interesującym projektem, jakim było zbudowanie ogromnego zbioru naturalnych obrazów opatrzonych etykietami. Etykieta obrazu opisuje jego zawartość i można ją wykorzystać, aby nauczyć modele AI poprawnego klasyfikowania obrazów. Zatem ten zbiór danych jest bezcennym narzędziem do trenowania modeli AI w zadaniach rozpoznawania obrazu. Dziś zbiór ImageNet zawiera ponad 14 milionów adnotowanych obrazów i miał kluczowe znaczenie w tworzeniu wielu aplikacji widzenia komputerowego.

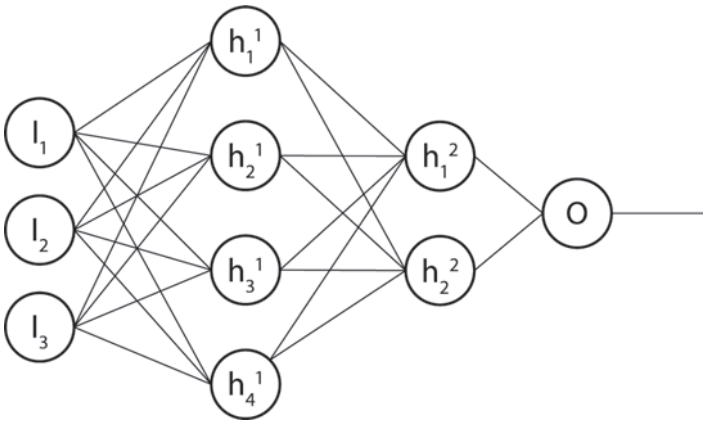
Co ciekawe, architektura AlexNet była podobna do konwolucyjnej sieci neuronowej (ang. *convolutional neural network*, CNN) zaproponowanej przez Yanna LeCuna w 1989 roku. Niestety w 1989 roku LeCun nie miał dostępu do zbioru ImageNet ani do procesora GPU o wydajności wystarczającej do wytrenowania głębokiego (wielowarstwowego) modelu. Jednak procesory GPU zdemokratyzowały dostęp do potężnych kalkulatorów, które mogą równolegle wykonywać tysiące operacji. Prawie każdy może sobie pozwolić na zakup GPU i skonfigurowanie systemu do trenowania sieci neuronowej. Eksperymenty można prowadzić w każdym laboratorium nawet bez dostępu do wyspecjalizowanych superkomputerów. W połączeniu z ImageNet i podobnymi zbiorami danych utworzonymi przez badaczy z pomocą internautów doprowadziło to do eksplozji w badaniach nad sieciami neuronowymi: zbadano najróżniejsze istniejące architektury i wymyślono nowe. AlexNet to CNN, a sieciom CNN przyjrzymy się w następnym rozdziale, kiedy będziemy szczegółowo omawiać widzenie komputerowe. Teraz zbadajmy model, od którego wszystko się zaczęło: perceptron wielowarstwowy.

ZDEJMOWANIE WARSTW

Wiesz już, jak działają pojedyncze neurony. Przedstawiłem ich historię i opowiedziałem, jak różne odkrycia zmieniły strukturę sztucznych neuronów. Pora więc wyjaśnić, jak funkcjonują sieci neuronowe. Chcemy zaprojektować sieć, która jest złożona z kilku warstw sztucznych neuronów, i zrozumieć, co dzieje się w każdym neuronie.

Rozważmy prosty MLP, nazywany też *w pełni połączoną* siecią neuronową. Możemy utworzyć prosty model z trzema węzłami wejściowymi, po których następuje warstwa z czterema neuronami, po której następuje warstwa z dwoma

neuronami, a na końcu znajduje się pojedynczy neuron wyjściowy (zob. rysunek 1.14). Warstwy pomiędzy wejściem i wyjściem — czyli sedno sieci neuronowej — nazywane są warstwami *ukrytymi*. Jak wspomniałem we wstępie, termin *uczenie głębokie*, albo *głębokie sieci neuronowe*, oznacza modele sieci neuronowych, które mają więcej niż jedną warstwę ukrytą. W tym przykładzie model ma dwie warstwy ukryte.

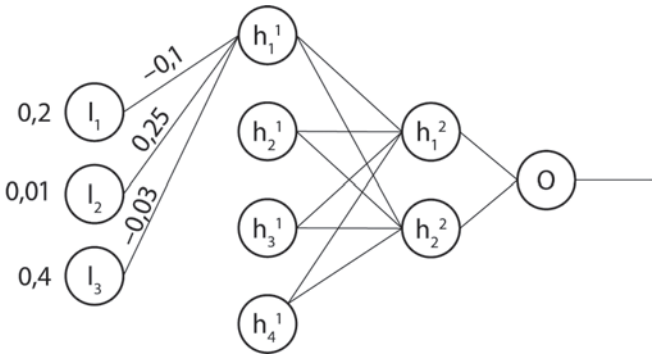


Rysunek 1.14. Perceptron wielowarstwowy (MLP) z dwiema warstwami ukrytymi. Trzy węzły wejściowe są połączone z pierwszą warstwą ukrytą (h^1), a warstwa h^1 jest połączona z drugą warstwą ukrytą (h^2), która jest połączona z neuronem wyjściowym

Pierwszą warstwę ukrytą, tę z czterema neuronami, będziemy nazywać h^1, a drugą warstwę ukrytą nazwiemy h^2. Sieci MLP są również znane jako w pełni połączone sieci neuronowe dlatego, że każdy neuron każdej warstwy jest połączony z każdym neuronem następnej warstwy. Węzeł wejściowy I_1 jest podłączony do h_1^1, h_2^1, h_3^1, h_4^1. Węzeł wejściowy I_2 również jest podłączony do h_1^1, h_2^1, h_3^1, h_4^1, podobnie jak węzeł wejściowy I_3. Wyjście węzła h_1^1 jest podłączone do węzłów h_1^2 i h_2^2. Wyjście węzła h_2^1 jest podłączone do węzłów h_1^2 i h_2^2, podobnie jak wyjścia węzłów h_3^1 i h_4^1. Wyjście węzła h_1^2 jest połączone z neuronem wyjściowym O, podobnie jak wyjście węzła h_2^2 (zob. rysunek 1.14). Wartość wyjściowa każdego neuronu zależy od wartości każdego związanego z nim połączenia i od funkcji aktywacji. Wartości związane z każdym połączeniem neuronowym nazywamy *wagami*, a funkcja aktywacji określa zakres wartości, które neuron może generować na wyjściu (lub w niektórych przypadkach może ograniczać wyjście neuronu do momentu przekroczenia określonego zakresu wartości wejściowych).

Celem funkcji aktywacji jest wprowadzenie nieliniowości do obliczeń wykonywanych przez sieć neuronową. Znaczenie nieliniowości omówię w dalszym punkcie, dotyczącym przestrzeni wektorowych. Na razie przypomnę tylko, że neurony biologiczne cechują się podobną nieliniowością. Hebb wykazał, że zależność między sygnałami wejściowymi na dendrytach a sygnałem wyjściowym na aksonie nie ma charakteru liniowego, tzn. wartość wyjściowa nie jest po prostu sumą sygnałów wejściowych. Sygnał wyjściowy neuronu jest regulowany przez pewną funkcję wewnętrzną, która uwzględnia sygnały wejściowe i przekształca je w ramach pewnego procesu. Sztuczne neurony w naszym modelu naśladują ten proces z użyciem matematycznych funkcji aktywacji. Badacze opracowali wiele funkcji aktywacji, a w zależności od problemu, który próbujemy rozwiązać, niektóre funkcje aktywacji sprawdzają się lepiej niż inne. Popularne funkcje aktywacji to *funkcja sigmoidalna* i *prostownik liniowy* (ang. *rectified linear unit*, ReLU). Funkcja sigmoidalna generuje wartości od 0 do 1, dlatego jest powszechnie stosowana wtedy, kiedy sieć neuronowa ma generować na wyjściu wartość prawdopodobieństwa; prawdopodobieństwa mieszczą się w przedziale od 0 do 1, gdzie 1 jest najwyższą możliwą wartością (szansą 100-procentową). Obecnie jedną z najpopularniejszych funkcji aktywacji jest funkcja ReLU. Sprawdza ona po prostu wartość wejściową i jeśli jest to wartość ujemna, zwraca 0, a jeśli jest to wartość dodatnia, zwraca wartość wejściową. W poniższym przykładzie używamy funkcji aktywacji ReLu dla wszystkich neuronów z wyjątkiem neuronu wyjściowego. Jest to typowe podejście w sieciach neuronowych przeznaczonych do *klasyfikacji binarnej* (klasyfikacja binarna oznacza rozdzielanie próbek danych na dwie możliwe klasy, np. kot kontra pies). W neuronie wyjściowym używamy sigmoidalnej funkcji aktywacji, żeby sygnał wyjściowy można było zinterpretować jako prawdopodobieństwo. Zobaczmy teraz, jak informacje przepływają przez sieć neuronową.

Najpierw obliczamy wyjście pierwszego neuronu w warstwie h^1. Przyjmijmy, że sieć neuronowa została już wytrenowana i mamy wartości wag dla każdego połączenia (zob. rysunek 1.15). Załóżmy, że wektor wejściowy, który chcemy przetworzyć, to I = [0,2, 0,01, 0,4].



Rysunek 1.15. Wartości wag połączeń z węzłem h_1^1. Dla przejrzystości połączenia z innymi neuronami w warstwie h^1 zostały usunięte

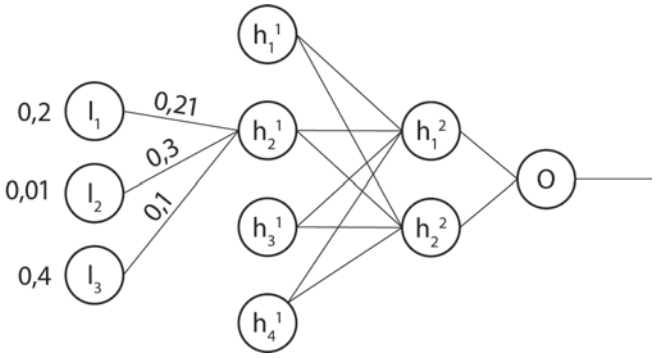
Obliczamy sumę ważoną na podstawie wektora wejściowego i wag połączeń:

$$h_1^1 = \text{ReLu}\,(w_{1,1}^1 I_1 + w_{1,2}^1 I_2 + w_{1,3}^1 I_3)$$
$$= \text{ReLu}\,(-0{,}1 \cdot 0{,}2 + 0{,}25 \cdot 0{,}01 + (-0{,}03 \cdot 0{,}4))$$

Jak zapewne pamiętasz, funkcja ReLu zwraca 0 dla wartości ujemnych.

$$= \text{ReLu}\,(-0{,}0295)$$
$$= 0$$

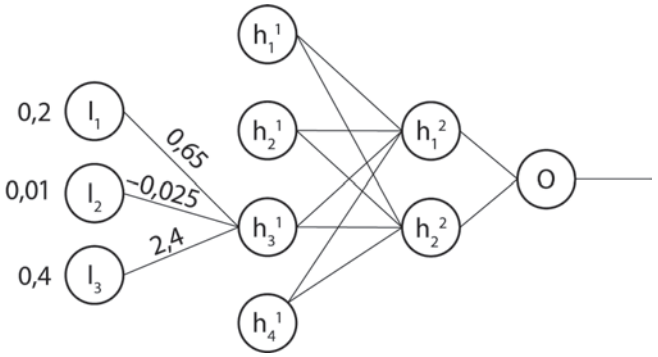
W ten sam sposób obliczamy wyjście drugiego neuronu w warstwie h^1 (zob. rysunek 1.16).



Rysunek 1.16. Wartości wag połączeń z węzłem h_2^1 (rysunek uproszczony, podobnie jak rysunek 1.15)

$$h_2^1 = \text{ReLu}\,(w_{2,1}^1 I_1 + w_{2,2}^1 I_2 + w_{2,3}^1 I_3)$$
$$= \text{ReLu}\,(0{,}21 \cdot 0{,}2 + 0{,}3 \cdot 0{,}01 + 0{,}1 \cdot 0{,}4)$$
$$= \text{ReLu}\,(0{,}085)$$
$$= 0{,}085$$

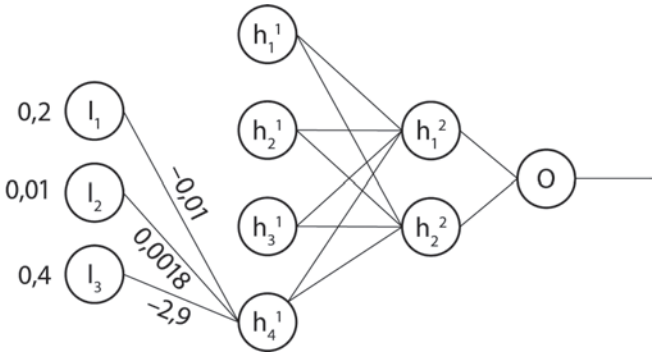
Wyjście trzeciego neuronu (zob. rysunek 1.17):



Rysunek 1.17. Wartości wag połączeń z węzłem h_3^1

$$h_3^1 = \text{ReLu}\,(w_{3,1}{}^1 I_1 + w_{3,2}{}^1 I_2 + w_{3,3}{}^1 I_3)$$
$$= \text{ReLu}\,(0{,}65 \cdot 0{,}2 + (-0{,}025) \cdot 0{,}01 + 2{,}4 \cdot 0{,}4)$$
$$= \text{ReLu}\,(1{,}1)$$
$$= 1{,}1$$

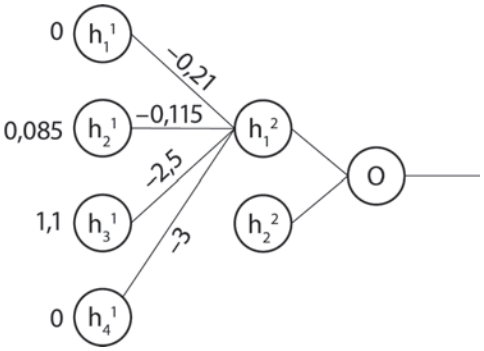
Wyjście czwartego neuronu (zob. rysunek 1.18):



Rysunek 1.18. Wartości wag połączeń z węzłem h_4^1

$$h_4^1 = \text{ReLu}\,(w_{4,1}{}^1 I_1 + w_{4,2}{}^1 I_2 + w_{4,3}{}^1 I_3)$$
$$= \text{ReLu}\,(-0{,}01 \cdot 0{,}2 + 0{,}0018 \cdot 0{,}01 + (-2{,}9) \cdot 0{,}4)$$
$$= \text{ReLu}\,(-1{,}l6)$$
$$= 0$$

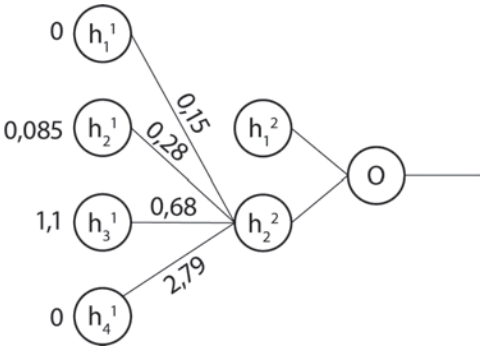
Teraz, kiedy mamy wyjście każdego neuronu z warstwy h^1, wartości te stają się wejściami warstwy drugiej, h^2 (zob. rysunek 1.19). W celu obliczenia wartości wyjściowej pierwszego neuronu z warstwy h^2 wykonujemy następujące działanie:



Rysunek 1.19. Wartości wejściowe i wagi połączeń z węzłem h_1^2. Zauważ, że każde wejście w warstwie h^2 jest wyjściem h^1 [0, 0,085, 1,1, 0] obliczonym w poprzednim etapie

$$h_1^2 = \text{ReLu}\,(w_{1,1}^2 h_1^1 + w_{1,2}^2 h_2^1 + w_{1,3}^2 h_3^1 + w_{1,4}^2 h_4^1)$$
$$= \text{ReLu}\,(-0{,}21 \cdot 0 + (-0{,}115) \cdot 0{,}085 + (-2{,}5) \cdot 1{,}1 + (-3) \cdot 0)$$
$$= \text{ReLu}\,(-2{,}75)$$
$$= 0$$

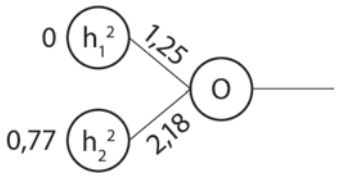
Wyjście drugiego neuronu w warstwie h^2 (zob. rysunek 1.20):



Rysunek 1.20. Wartości wejściowe i wagi połączeń z węzłem h_2^2

$$h_2^2 = \text{ReLu}\,(w_{2,1}^2 h_1^1 + w_{2,2}^2 h_2^1 + w_{2,3}^2 h_3^1 + w_{2,4}^2 h_4^1)$$
$$= \text{ReLu}\,(0{,}15 \cdot 0 + 0{,}28 \cdot 0{,}085 + 0{,}68 \cdot 1{,}1 + 2{,}79 \cdot 0)$$
$$= \text{ReLu}\,(0{,}77)$$
$$= 0{,}77$$

Wyjście sieci neuronowej oblicza się z użyciem wyjść warstwy h^2 jako wejść neuronu wyjściowego (zob. rysunek 1.21).



Rysunek 1.21. Wyjście warstwy h^2 jako wejście warstwy wyjściowej oraz wagi połączeń z warstwą wyjściową

Zauważ, że w tym przykładzie w warstwie wyjściowej używamy sigmoidalnej funkcji aktywacji. Funkcja sigmoidalna ma następującą postać: $\frac{1}{1+e^{-(x)}}$.

$O = \text{sigmoid}\,(w_{1,1}{}^3h_1{}^2 + w_{1,2}{}^3h_2{}^2)$
$= \text{sigmoid}\,(1{,}25 \cdot 0 + 2{,}18 \cdot 0{,}77)$
$= \text{sigmoid}\,(1{,}68)$
$= \frac{1}{1+e^{-(1{,}68)}}$
$= 0{,}84$

To wszystko! Zaczęliśmy od wartości wejściowej i zbioru wstępnie określonych wag, po czym przetworzyliśmy wejście z użyciem sieci neuronowej i uzyskaliśmy wynik. Sigmoidalna funkcja aktywacji jest nieco skomplikowana, ale nie musisz się tym zbytnio przejmować. Wystarczy wiedzieć, że przyjmuje ona wartość wejściową i zwraca wartość wyjściową z zakresu od 0 do 1, którą można traktować jak prawdopodobieństwo. Na tym etapie nie wiesz jeszcze, jak interpretować tę wartość wyjściową, ale nic nie szkodzi. Celem tego ćwiczenia było zademonstrowanie operacji odbywających się wewnątrz sieci neuronowej. Choć na razie trudno powiedzieć, co znaczy 0,84 i dlaczego może to być przydatne, wiesz już, jak uzyskuje się ten wynik, i widzisz, że są to proste działania. W poszczególnych obliczeniach nie ma żadnej magii — są to głównie działania dodawania i mnożenia. Te dodawania i mnożenia z użyciem wartości wejściowych i wag połączeń neuronowych nazywamy operacją sumy ważonej i właśnie ona jest paliwem, które napędza prognozy w sztucznych sieciach neuronowych.

W potocznym języku wynik działania sieci neuronowej nazywamy *przewidywaniem* albo *prognozą*. Wynika to stąd, że w ogólnym sensie sieć neuronowa generuje wartość, która jest poprawna z pewnym prawdopodobieństwem. Na podstawie wcześniejszych danych zaobserwowanych podczas treningu sieć neuronowa buduje pewien model świata. Kiedy zaprezentujemy jej nowe dane wejściowe, generuje prognozę. W przypadku klasyfikacji prognoza jest zazwyczaj prawdopodobieństwem, że dane wejściowe należą do określonej klasy obiektów. Jednak współczesnych sieci neuronowych używa się nie tylko do klasyfikacji — innym zastosowaniem są *problemy regresyjne* (rodzaj problemu prognostycznego), w których sieć neuronowa przewiduje wynik liczbowy dla pewnych danych wejściowych.

Wiesz już, jak wygląda wykonywanie sieci neuronowej, a zatem wskoczyłeś na głęboką wodę. Teraz powstaje pytanie: jak wykorzystać te narzędzia do rozwiązywania rzeczywistych problemów? Aby na nie odpowiedzieć, wrócimy na suchy ląd i stopniowo będziemy zanurzać się coraz głębiej, rozważając kilka przykładów. Najpierw przyjrzymy się przykładom klasyfikacji, a następnie przykładowi regresji.

ZASTOSOWANIA KLASYFIKACJI

Przypuśćmy, że otrzymaliśmy następujące zadanie: studio hollywoodzkie chce, żebyśmy opracowali narzędzie, które będzie skanować recenzje filmów publikowane przez użytkowników jakiegoś forum dyskusyjnego i określać, czy są one pozytywne, czy negatywne. Mamy trochę danych — w tym przypadku kilka wierszy tekstu, w których recenzent opisuje swoje wrażenia — i musimy stworzyć narzędzie, które przeanalizuje te dane i powie nam, czy recenzja jest pochlebna, czy nie.

Jeśli przez chwilę pomyślisz o tym problemie, uświadomisz sobie, że nie jest on tak prosty, jak początkowo mogłoby się wydawać. Naiwnym rozwiązaniem byłoby szukanie pewnych słów kluczowych — *zły*, *dobry*, *w porządku* — i określanie wymowy recenzji na podstawie występowania tych słów. Jeśli jednak przyjmiemy takie proste podejście, szybko zauważymy, że problem jest bardziej zniuansowany. Rozważ następujące zdanie: „Obawiałam się, że ten film będzie okropny. Okazało się, że wcale nie był zły". Recenzja ta jest niewątpliwie pozytywna, ale zawiera dwa negatywne słowa: *okropny* i *zły*. A teraz rozważ drugie

zdanie: „Najwspanialsze w tym filmie było to, że wreszcie się skończył. Nawet najlepsi aktorzy by go nie uratowali". To przykład negatywnej recenzji, która zawiera dwa pozytywne słowa: *najwspanialsze* i *najlepsi*. Jest oczywiste, że nie możemy po prostu przyglądać się pojedynczym słowom, lecz musimy wziąć pod uwagę cały tekst oraz związki między słowami.

Problemy tego typu należą do dziedziny *przetwarzania języka naturalnego* (ang. *natural language processing*, NLP), a sieci neuronowe są najlepszymi modelami, jakimi dziś dysponujemy, jeśli chodzi o rozwiązywanie problemów NLP. Kiedy prosisz Aleksę o zamówienie nowej pary skarpet albo każesz asystentowi Google Home odtworzyć inną piosenkę, neuronowa sieć NLP przetwarza Twoją mowę i interpretuje Twoje polecenia. Jak więc rozwiążemy problem recenzji filmowych? Jak nauczymy sieć, co sprawia, że recenzja jest pozytywna?

Pierwsze, co musimy zrobić, to pozyskać treningowy zbiór danych. W tym przypadku danymi treningowymi jest po prostu duży zbiór recenzji filmowych, które możemy zaprezentować sieci neuronowej, aby nauczyć ją, jak wyglądają recenzje pozytywne, a jak negatywne. Zbiór treningowy zwykle zawiera tysiące próbek — w naszym przykładzie próbką jest pojedyncza recenzja. Zbiór treningowy zawiera też szczególny rodzaj informacji zwany *etykietami*. Ponieważ chcemy wytrenować model tak, aby odróżniał recenzje pozytywne od negatywnych, próbki muszą zawierać etykiety klasyfikacyjne, które określają każdą recenzję jako pozytywną lub negatywną. Etykiety te są zazwyczaj przygotowywane przez ludzi — badaczy i ochotników — którzy żmudnie etykietują tysiące próbek w treningowych zbiorach danych.

Zostaliśmy zatrudnieni do stworzenia narzędzia, które potrafi klasyfikować recenzje filmowe. Konstruujemy sieć neuronową, która przyjmuje dane wejściowe w postaci fragmentów tekstu i generuje klasyfikację: recenzja pozytywna lub negatywna. Musimy znaleźć treningowy zbiór danych, który wykorzystamy do wytrenowania nowo utworzonego modelu. Na szczęście Uniwersytet Stanforda udostępnia zbiór opatrzonych etykietami recenzji pozyskanych z witryny IMDb (Internet Movie Database). Zbiór danych, dostępny bezpłatnie dla każdego, zawiera 50 000 próbek recenzji podzielonych na 25 000 próbek treningowych i 25 000 próbek testowych. Zbiory danych treningowych zwykle dzielą się na próbki treningowe i testowe. Kiedy trenujemy sieć neuronową, chcemy, żeby uczyła się na przykładowych danych, ale nie „na pamięć". Jeśli sieć nauczy się poprawnej odpowiedzi na każdy przykład treningowy poprzez zapamiętanie

próbek treningowych, nie powie nam to zbyt wiele o tym, jak będzie sobie radzić w rzeczywistych sytuacjach. Próbki testowe pozwalają nam ocenić, czy sieć naprawdę nauczyła się informacji, które można zastosować do wcześniej nie widzianych danych, czy też po prostu zapamiętała dane treningowe.

Jak widać w powyższym prostym przykładzie, sieci neuronowe to systemy matematyczne, które operują na liczbach. Oczekują liczb jako danych wejściowych, przekształcają te liczby z użyciem operacji matematycznych i generują liczby jako dane wyjściowe. Częścią projektowania modelu sieci neuronowej jest wymyślenie, jak reprezentować, czy też *kodować*, dane jako liczby. Przypuśćmy, że mamy następującą recenzję: „Ten film był niewiarygodnie dobry". Jak przekształcić to zdanie w zbiór liczb? Informatycy i statystycy wymyślili na to kilka sposobów.

Jednym ze sposobów jest stworzenie słownika wyrazów używanych w recenzjach ze zbioru treningowego. Tworzymy listę wszystkich słów użytych we wszystkich recenzjach w zbiorze danych. Następnie identyfikujemy najczęściej używane słowa. Przypuśćmy, że nasza lista liczy 80 000 słów; możemy wybrać 10 000 słów najczęściej występujących w recenzjach i utworzyć słownik liczący 10 000 słów. Każde słowo w tym słowniku jest unikatowe, tzn. występuje na liście tylko raz, i z każdym związany jest indeks. Pierwsze słowo ma indeks 1, drugie ma indeks 2 itd. Przypomnijmy sobie zdanie, które chcemy przeanalizować: „Ten film był niewiarygodnie dobry". Możemy zamienić je w wektor: [Ten, film, był, niewiarygodnie, dobry]. Następnie sprawdzamy ich indeksy na liście 10 000 słów i zastępujemy słowa w wektorze ich indeksami. Kiedy jednak to robimy, zauważamy, że słowa *niewiarygodnie* nie ma na utworzonej przez nas liście 10 000 słów. Oznacza to, że nie występowało ono wystarczająco często w treningowym zbiorze danych i nie figuruje w słowniku, więc możemy je odrzucić. Wektor przyjmuje więc postać [Ten, film, był, dobry]. Po podstawieniu indeksów każdego słowa wektor przyjmuje postać [10, 4, 22, 100]. Możemy zinterpretować to następująco: *ten* jest 10. słowem na liście, *film* jest 4. słowem, *był* jest 22. słowem, a *dobry* jest 100. słowem. Udało się nam przekształcić recenzję w zbiór liczb, co jest konieczne, ponieważ sieć neuronowa potrzebuje liczb do przetwarzania informacji. Jesteśmy na dobrej drodze, żeby poprosić model o przewidzenie, czy ta próbka recenzji jest pozytywna, czy negatywna. Nie możemy jeszcze jednak tego zrobić, musimy dokonać jeszcze jednej modyfikacji.

Ponieważ sieci neuronowe pracują z liczbami, chcemy, żeby zakres wartości wszystkich danych wejściowych był mniej więcej taki sam — nazywa się to *normalizowaniem* danych. Przypuśćmy, że mamy zbiór recenzji, które używają słów znajdujących się na początku naszego słownika, tzn. wszystkie słowa w tych recenzjach mają indeksy mniejsze niż 100. W wektorach wyrazowych reprezentujących te recenzje wszystkie wartości byłyby mniejsze od 100. Przypuśćmy teraz, że następny zbiór recenzji składa się głównie ze słów, które znajdują się na końcu naszego słownika; powiedzmy, że wektory składają się z wartości większych niż 8000 (np. [8001, 9245, 8444, 9001]). Przypomnij sobie, jak dane wejściowe są przetwarzane przez neurony w sieci. Bierzemy wartości wejściowe i przeprowadzamy operację sumy ważonej z uwzględnieniem wag połączeń neuronu, a wynik tej operacji jest modulowany przez funkcję aktywacji neuronu. Wysokie i niskie wartości wprowadzane do neuronu inaczej wpływają na wyjście tego neuronu. Nie chcemy, aby wyjście sieci zależało od tego, z której części listy pochodzą słowa użyte w recenzji, bo informacja ta może nie mieć nic wspólnego z rzeczywistą *wymową* recenzji. Chcemy, aby sieć nauczyła się, co sprawia, że recenzja jest pozytywna albo negatywna. Wymaga to, aby sieć nauczyła się semantycznego znaczenia każdego słowa oraz relacji między wyrazami w słowniku. Pozycje wyrazów na liście słownikowej zostały jednak wybrane arbitralnie. Tworząc słownik złożony z 10 000 wyrazów, nie organizowaliśmy słów w żaden znaczący sposób, nie chcemy zatem, aby sieć nauczyła się inaczej ważyć słowa pochodzące z różnych części listy, do czego z pewnością by doszło, gdyby niektóre indeksy były znacznie większe od innych.

Jeden ze sposobów normalizowania wektora wejściowego tak, aby wartości dla wszystkich słów należały do tego samego zakresu, jest znany jako *kodowanie z gorącą jedynką* (ang. *one-hot encoding*). Zamiast 4-wymiarowego (zawierającego cztery wartości) wektora [10, 4, 22, 100] dla zdania „Ten film był dobry" tworzymy wektor mający 10 000 wymiarów, pod odpowiednimi indeksami umieszczając w nim 1 dla każdego słowa, które pojawiło się w recenzji, i 0 dla słów, które nie pojawiły się w recenzji. W ten sposób powstaje wektor 10 000 wartości, który wygląda tak: [0, 0, 0, 1, 0, 0, 0, 0, 0, 1, … (jedenaście zer), 1, … (siedemdziesiąt siedem zer), 1, 0, 0, 0, … (zera aż do 10 000. miejsca)]. Wydaje się to skomplikowane, ale wcale takie nie jest. Umieściliśmy 1 w 4. miejscu wektora, 1 w 10. miejscu, 1 w 22. miejscu i 1 w 100. miejscu. We wszystkich pozostałych miejscach umieściliśmy 0. Teraz mamy metodę kodowania recenzji jako wektorów wartości, które możemy przekazywać do sieci neuronowej do analizy.

Wektor ten koduje informacje o słowach w recenzji, ale nie przypisuje większych wag grupom słów w zależności od ich położenia na liście. Wektor jest znormalizowany, a wszystkie jego wartości są równe 0 lub 1.

Kodowanie z gorącą jedynką ma tę zaletę, że wszystkie wektory wejściowe — czyli wszystkie recenzje — będą miały ten sam rozmiar. Recenzje mogą być dłuższe lub krótsze, ale opisany wyżej proces generowania wektora wejściowego będzie tworzył wektory o jednakowej długości: 10 000 wartości. Jak się niebawem przekonasz, jest to ważne, ponieważ nasza sieć neuronowa musi wiedzieć, ile wartości jest w danych wejściowych, a liczba wartości nie może się zmieniać w zależności od próbki.

Wnikliwy Czytelnik może zauważyć kilka potencjalnych problemów związanych z tą metodą kodowania danych wejściowych. Wybierając słownik, który jest znacznie mniejszy od liczby unikatowych wyrazów w zbiorze danych treningowych (i ogólnie w języku polskim), godzimy się na to, że niektóre słowa pojawiające się w recenzjach będą odrzucane. Są to słowa, które pojawiają się najrzadziej w zbiorze danych treningowych. Skąd mamy wiedzieć, że nie odrzucamy wartościowych informacji? Co więcej, opisana metoda kodowania ma pewną wadę. Załóżmy, że w recenzji trzykrotnie pojawia się słowo *dobry*. W proponowanej metodzie kodowania z gorącą jedynką każde słowo możemy zakodować tylko raz, ponieważ na liście jest tylko jeden indeks dla słowa *dobry*. Na szczęście problem, który staramy się rozwiązać, polega na klasyfikowaniu recenzji jako negatywnej lub pozytywnej. Nie staramy się określić stopnia pozytywności lub negatywności (ponieważ nas o to nie poproszono). Na przykład model nie musi klasyfikować recenzji jako „dobrych", „świetnych", „najlepszych" czy też „złych", „fatalnych" lub „najgorszych". Oznacza to, że utrata informacji o tym, ile razy użytkownik stwierdził, że film jest „dobry", może nie być aż tak istotna. Jednak, ogólnie rzecz biorąc, są to uzasadnione obawy. Istnieją inne, bardziej złożone metody kodowania informacji, które pozwalają uniknąć niektórych z tych problemów i umożliwiają kodowanie częstotliwości słów. Byłyby one niezbędne w klasyfikowaniu wieloklasowym, w którym próbujemy przewidywać stopień odczuć, a nie tylko binarny przypadek pozytywna/negatywna. Jednak nawet pomimo wspomnianych wad metoda ta działa całkiem nieźle i może dawać naprawdę dobre wyniki w zadaniach klasyfikacji binarnej, takich jak to, które tu omawiam.

No dobrze, mamy teraz wektor wejściowy. Co dalej? Przedstawiamy go modelowi sieci neuronowej. Zdefiniujmy architekturę modelu sieci neuronowej zdolną do rozwiązania tego problemu. Możemy zaprojektować sieć neuronową składającą się z warstwy wejściowej, dwóch warstw ukrytych i warstwy wyjściowej. Warstwa wejściowa będzie przyjmować 10 000 wartości. Pierwsza warstwa ukryta będzie składać się z 32 neuronów z funkcją aktywacji ReLU. Druga warstwa ukryta będzie składać się z kolejnych 32 neuronów, również z funkcją aktywacji ReLU. Czy domyślasz się, ile neuronów będzie potrzebnych w warstwie wyjściowej? Warstwa wyjściowa będzie składać się z 1 neuronu z sigmoidalną funkcją aktywacji. Wybór sigmoidalnej funkcji aktywacji dla neuronu wyjściowego jest istotny, ponieważ chcemy, żeby wynik był prawdopodobieństwem, tzn. żeby wyjście sieci wskazywało prawdopodobieństwo, że recenzja jest pozytywna lub negatywna, a funkcje sigmoidalne zwracają wartości z zakresu od 0 do 1, co można zinterpretować jako prawdopodobieństwo. Jeśli nie jest to jeszcze całkiem jasne, nie przejmuj się, bo niebawem wyjaśnię to dokładniej.

Przeanalizujmy architekturę sieci neuronowej. Wybraliśmy funkcję aktywacji ReLU dla neuronów w warstwach ukrytych. Funkcja ReLU jest prosta: jeśli jej wartość wejściowa jest ujemna, zwraca 0, a jeśli wartość wejściowa jest dodatnia, zwraca tę samą wartość (tzn. wartość wejściową). Funkcji ReLU używamy dlatego, że w większości architektur sztucznej sieci neuronowej i w większości obecnych zastosowań badacze ustalili empirycznie (metodą prób i błędów), że działa ona najlepiej. Przez długi czas bardzo popularne były funkcje sigmoidalne, ale funkcja ReLU ma pewne własności, które ułatwiają proces treningu, dlatego z czasem zaczęła być stosowana przez większość badaczy. Bardziej interesującym pytaniem jest to, dlaczego funkcja ReLU działa tak dobrze, ale niestety odpowiedź nie jest do końca jasna. Jak powiedział kiedyś słynny badacz sztucznej inteligencji Geoffrey Hinton, „to wszystko jest zmyślone".

Część tego, co robimy w sztucznych sieciach neuronowych, jest inspirowana systemami biologicznymi — jak wiadomo, neurony biologiczne mają wewnętrzne stany, które sprawiają, że generują one sygnały wyjściowe na podstawie sygnałów wejściowych, ale w sposób nieliniowy. Oznacza to, że siła sygnału wyjściowego nie zawsze jest proporcjonalna do siły sygnału wejściowego. ReLU, sigmoid i inne funkcje matematyczne pomagają dodać nieliniowość do sztucznych neuronów, ale podobieństwa do systemów biologicznych na tym się kończą. Nie rozumiemy jeszcze wystarczająco systemów biologicznych, aby opracować

teorię, która pozwoliłaby nam znaleźć odpowiednią funkcję aktywacji dla sztucznego neuronu. Badacze empirycznie wypróbowują różne koncepcje i zachowują te, które się sprawdziły. Dobieranie rozmiaru ukrytych warstw — liczby neuronów w każdej warstwie — również jest bardziej sztuką niż nauką. Nie ma zbioru reguł, które określałyby konkretną liczbę neuronów dla danego problemu. Przez dziesięciolecia badań zbudowano pewną intuicję. Istnieje korelacja między rozmiarem danych wejściowych, rozmiarem zbioru danych treningowych i zakresem liczby neuronów, których należy użyć, ale żaden wzór nie powie nam, jak zaprojektować sieć. Znaczna część naszej wiedzy ma charakter empiryczny. Badacze wypróbowują różne architektury i testują swoje modele, a następnie modyfikują je, jeśli system nie działa tak, jak powinien.

Zaprojektowaliśmy model sieci neuronowej i proces służący do przekształcania danych wejściowych w liczby, które możemy przekazać do sieci neuronowej. W warstwie wyjściowej zastosowaliśmy sigmoidalną funkcję aktywacji, ponieważ chcemy interpretować wynik jako prawdopodobieństwo. Jak to dokładnie działa? Jak interpretujemy dane wyjściowe zbudowanej przez nas sieci neuronowej? Przypomnijmy, że celem naszej sieci jest przypisywanie sygnału wejściowego — recenzji filmu — do jednej z dwóch klas: pozytywnej lub negatywnej. Jak już wiesz, sieci neuronowe rozumieją tylko liczby. Dlatego musimy zakodować klasy „pozytywną" i „negatywną" za pomocą liczb. Możemy to zrobić przez przypisanie klasie „pozytywnej" etykiety 1, a klasie „negatywnej" etykiety 0. Moglibyśmy też postąpić odwrotnie: przyjąć, że klasa „negatywna" ma wartość 1, a klasa „pozytywna" ma wartość 0. Najważniejsze to zachować konsekwencję. Kiedy sieć neuronowa generuje na wyjściu wartość mieszczącą się w przedziale od 0 do 1 — ze względu na użycie funkcji sigmoidalnej — interpretujemy tę wartość jako poziom ufności, że dane wejściowe należą do klasy 1 lub 0. Załóżmy na przykład, że sieć generuje wartość 0,78, a my przyjęliśmy, że etykieta 1 oznacza recenzję pozytywną. Wynik ten mówi nam, że sieć neuronowa ma 78% pewności, że recenzja jest pozytywna. Oznacza to, że na podstawie danych treningowych istnieje 78-procentowe prawdopodobieństwo, że recenzja, którą właśnie przeanalizowała sieć, jest recenzją pozytywną. Podobnie wynik 0,5 oznacza, że istnieje 50% szans, że recenzja jest pozytywna.

Zastanówmy się przez chwilę nad znaczeniem tego, co udało nam się osiągnąć. Wzięliśmy wejściową sekwencję tekstu, przekształciliśmy ją w format, który można przeanalizować za pomocą modelu matematycznego — sieci neuronowej —

ustaliliśmy, jak zaprojektować sieć neuronową, która potrafi przypisywać tekst do różnych kategorii na podstawie przekazu emocjonalnego (typ klasyfikacji, w którym próbujemy określić nastrój autora tekstu, nazywany jest także *analizą odczuć*), i nauczyliśmy się interpretować dane wyjściowe sieci neuronowej. Proces rozwiązywania tego problemu może nie jest intuicyjny, ale zaskakująco prosty, kiedy rozłożymy go na podstawowe operacje. To tylko mnożenia i dodawania z odrobiną funkcji aktywacyjnych.

Zadanie analizy odczuć było przez długi czas bardzo trudnym problemem. Jak widziałeś, tekst nie jest prostą sekwencją, w której każde słowo ma tę samą wagę. Niektóre słowa mają większe znaczenie dla przekazywania wymowy zdania, a między poszczególnymi wyrazami mogą zachodzić złożone relacje (np. słowo znajdujące się na początku zdania może podkreślać znaczenie słowa znajdującego się dużo dalej). Weźmy takie zdanie: „To interesujące, że w niektórych krajach, zwłaszcza w rejonach tropikalnych, deszcz pada przez większość roku". Słowo „interesujące" ma ścisły związek z „deszczem" i z faktem, że pada on przez „większość roku". Zauważ, ile słów musieliśmy pominąć, aby dotrzeć do znaczącej części zdania. A teraz takie zdanie: „Trudno wyjaśnić, co sprawia, że film jest interesujący". W tym przypadku relacja jest bardzo bliska. Słowo „interesujący" jest bezpośrednio powiązane ze swoim najbliższym sąsiadem, wyrazem „film". Z powodu tych niuansów trudno jest opracować zbiór sztywnych reguł o postaci „jeśli taki wzorzec, to taki wybór", jakich używano w klasycznych inteligentnych systemach. Sieci neuronowe są tak skuteczne dlatego, że nie musimy im tłumaczyć, co sprawia, że recenzja jest pozytywna lub negatywna — co jest niełatwe, bo trudno określić ogólne reguły dotyczące tych problemów. (Możesz przekonać się o tym samodzielnie: spróbuj zapisać zbiór reguł opisujących, co sprawia, że zdanie jest pozytywne, a następnie zastosować te reguły do zbioru przypadkowych postów). Natomiast sieci neuronowe same uczą się identyfikować te reguły. Imponujące!

Analizując proces wykonywania sieci neuronowej oraz ucząc się interpretowania jej wejść i wyjść, pominęliśmy bardzo ważne zagadnienie: trening. W tym rozdziale wyjaśnię go skrótowo. Następnie w kolejnych dwóch rozdziałach będziesz stopniowo poszerzać wiedzę i szczegółowo przyjrzymy się procesowi treningu. Jak widzisz, proces wykonywania sieci neuronowej jest dość prosty, a używana w nim matematyka ledwo zahacza o poziom szkoły średniej. Proces treningu to zupełnie inna historia.

Dlaczego w ogóle potrzebujemy treningu? W dotychczasowych przykładach używaliśmy wag połączeń między neuronami sieci tak, jakby zostały w magiczny sposób ustawione na odpowiednie wartości, aby dostarczać poprawne wyniki. Niestety sieci neuronowe nie zaczynają swojego życia w taki sposób. Początkowo wagi połączeń są inicjalizowane losowo, więc przewidywania sieci są w dużej mierze niedokładne. Dopiero w procesie treningu stopniowo reguluje się wagi, aż w końcu prognozy zaczynają mieć sens. Wagi modelu reprezentują wyuczone informacje. Zaczynamy od losowo zainicjalizowanych wag oraz zbioru danych treningowych. Zbiór danych to zbiór próbek z dziedziny problemu. Jeśli na przykład chcemy nauczyć sieć neuronową odróżniać jabłka od bananów, zbiór danych musi zawierać zróżnicowane zdjęcia jabłek i bananów. Jeśli chcemy, aby sieć neuronowa odróżniała recenzje pozytywne od negatywnych, zbiór danych musi zawierać próbki recenzji pozytywnych i negatywnych. Podczas treningu sieć neuronowa otrzymuje próbkę ze zbioru danych treningowych.

W przykładzie z klasyfikowaniem obrazów próbką treningową jest jeden obraz ze zbioru danych; w przykładzie z analizą odczuć, takim jak nasz problem klasyfikowania recenzji filmowych, próbką treningową jest jedna recenzja ze zbioru danych. Przedstawiamy próbkę treningową sieci neuronowej, która przewiduje wynik. Ponieważ sieć nie jest jeszcze wytrenowana, prognoza będzie niedokładna — sieć może na przykład stwierdzić, że bardzo pozytywna recenzja jest negatywna. Jako że jest to próbka treningowa, towarzyszy jej etykieta, która wskazuje jej rzeczywistą klasę: recenzje pozytywne są oznaczone etykietą „pozytywna", a recenzje negatywne — etykietą „negatywna" (pamiętaj, że etykiety są liczbami, więc 1 to etykieta „pozytywna", a 0 to etykieta „negatywna"). Następnie używamy wzoru matematycznego, aby ustalić, jak bardzo błędne są bieżące przewidywania. Mówiąc ściślej, obliczamy, jak bardzo prognozy sieci neuronowej różnią się od rzeczywistych etykiet, po czym wykorzystujemy wynik tych obliczeń do wyregulowania wag modelu w taki sposób, że kiedy model ponownie zobaczy tę próbkę, to nieco bardziej zbliży się do prawdy. Podczas treningu przeprowadzamy tę operację dla każdej próbki w zbiorze danych i przez pewną liczbę *epok* (tzn. cyklów, w których sieć neuronowa widzi raz każdą próbkę). Proces treningu zwykle zajmuje wiele epok, więc model wielokrotnie przetwarza każdą próbkę ze zbioru treningowego, za każdym razem wydobywając z niej nieco więcej informacji i przenosząc tę wiedzę do wag sieci.

Przyjrzyjmy się innemu przykładowi klasyfikacji, tym razem z dziedziny wizji. Na początku lat 90. w Stanach Zjednoczonych intensywnie pracowano nad stworzeniem algorytmów do rozpoznawania pisanych odręcznie cyfr na kopertach pocztowych. W tym przykładzie zbiór danych treningowych składa się z 50 000 pisanych odręcznie cyfr. Ten problem klasyfikacji ma dziesięć klas, ponieważ potrzebujemy modelu, który nauczy się rozróżniać dziesięć cyfr: od 0 do 9. Obrazy pisanych odręcznie cyfr w zbiorze danych mają rozdzielczość 28×28 pikseli. W przykładzie analizy odczuć przekształcaliśmy próbki recenzji w wektory, w których każde słowo było innym wymiarem wektora. Oznacza to, że dla recenzji składającej się z 10 słów tworzyliśmy wektor o 10 wartościach, który następnie przekształcaliśmy w wektor kodowania z gorącą jedynką zawierający 10 000 wartości. W przypadku klasyfikacji obrazu każdy piksel jest innym wymiarem wektora wejściowego, więc dla obrazów o wymiarach 28×28 pikseli tworzymy wektor o 784 wartościach ($28 \cdot 28 = 784$), w którym każda wartość reprezentuje jasność piksela: wartość 0 oznacza czarny piksel, wartość 1 oznacza biały piksel, a każda wartość pośrednia reprezentuje skalę szarości. Teraz, gdy zdefiniowaliśmy wektor wejściowy, możemy przystąpić do projektowania sieci neuronowej klasyfikującej pisane odręcznie cyfry.

Zaczynamy od warstwy wejściowej złożonej z 784 węzłów, po jednym na każdy piksel wektora wejściowego. Następnie dodajemy warstwę 512 neuronów z aktywacją ReLU, a potem warstwę 64 neuronów również z aktywacją ReLU. Na koniec dodajemy warstwę wyjściową złożoną z 10 neuronów z funkcją regresji softmax. Wybór warstw zawierających 512 i 64 neurony ponownie jest kwestią intuicji oraz prób i błędów. Zazwyczaj sprawdzamy kilka projektów, aż znajdziemy taki, który jest efektywny (pod względem liczby parametrów) i działa dobrze.

Nie omówiłem jeszcze funkcji regresji softmax. Możesz myśleć o niej jak o funkcji sigmoidalnej, ale przeznaczonej do problemów wieloklasowych. Podczas gdy sigmoidalnej funkcji aktywacji używa się do problemów klasyfikacji binarnej w celu obliczenia prawdopodobieństwa, że dane wejściowe należą do jednej klasy — kot kontra pies, jabłko kontra banan — funkcje regresji softmax przekształcają wynik sieci neuronowej w prawdopodobieństwo w taki sposób, że wszystkie ich wyjścia sumują się do wartości 1. Na przykład nasza sieć ma 10 wyjść, po jednym dla każdej możliwej klasy cyfry wejściowej: od 0 do 9. Dla danego obrazu wejściowego sieć może wygenerować na wyjściu następujące 10 wartości: 0,80, 0,1, 0,0125, 0,0125, 0,005, 0,0025, 0,0168, 0,0168, 0,017, 0,0169.

Wynik ten mówi nam, że sieć ma 80% pewności, iż wartość wejściowa reprezentuje 0, 10% pewności, że reprezentuje 1, 1,25% pewności, że reprezentuje 2, itd. Jeśli zsumujemy wszystkie wartości wyjściowe, otrzymamy 1. Możemy trenować tę sieć neuronową na 50 000 obrazów treningowych przez pięć epok. Oznacza to, że sieć przetworzy zbiór danych treningowych pięć razy, ucząc się przewidywać poprawną etykietę dla każdego obrazu. Po mniej więcej pięciu epokach nauki sieć neuronowa może osiągnąć mniej więcej 98-procentową dokładność prognoz, co oznacza, że będzie poprawnie przewidywać klasę cyfry wejściowej w 98% przypadków. Jest to godne uwagi, biorąc pod uwagę, że przed pojawieniem się sieci neuronowych żaden inny algorytm rozpoznawania obrazu nawet nie zbliżał się do takiego poziomu dokładności.

Dlaczego taki algorytm jest użyteczny? W sensie praktycznym można wykorzystać go do sortowania listów na poczcie. Moglibyśmy zbudować linię maszyn sortujących, które otrzymywałyby długi strumień kopert. Maszyny przyglądałyby się kopertom, rozpoznawały kod pocztowy i sortowały listy w zależności od regionów dystrybucji. Jest też użyteczny w innym sensie: pokazuje nam, że można względnie łatwo budować systemy, które potrafią interpretować obrazy. Przed erą sieci neuronowych systemy wizyjne uważano za najtrudniejszy obszar automatyzacji, więc opracowanie algorytmu, który potrafi rozpoznać i zinterpretować obraz, było ogromnym osiągnięciem. Dało badaczom przekonanie, że widzenie komputerowe jest osiągalnym celem. Więcej o widzeniu komputerowym i zaawansowanych algorytmach wizyjnych powiem w następnym rozdziale, przy okazji odkrywając nową architekturę sieci neuronowej, która rozwiązuje kluczowy problem naszej skromnej sieci MLP. W przypadku obrazów liczących 28×28 pikseli (bardzo niska rozdzielczość, niepraktyczna w przypadku większości rzeczywistych zastosowań widzenia komputerowego) potrzebowaliśmy sieci neuronowej z 784 neuronami wejściowymi, co z kolei wymaga zastosowania mniej więcej takiej samej liczby neuronów w kolejnych warstwach. Wyobraź sobie rozmiar sieci neuronowej do przetwarzania obrazów o rozdzielczości kilku tysięcy pikseli! Przetwarzanie obrazów za pomocą sieci MLP szybko staje się trudne, ponieważ potrzebujemy coraz więcej neuronów, przez co radykalnie rosną wymagania pamięciowe. Kiedy będę omawiać widzenie komputerowe, poznasz inny typ sieci, która rozwiązuje problem wizji przy znacznie mniejszym zużyciu zasobów.

ZASTOSOWANIA REGRESJI

Problemy klasyfikacji, takie jak analiza odczuć i rozpoznawanie obrazów, są jednym z typowych zastosowań sieci neuronowych. Inne popularne zastosowanie, problemy regresji, polega na przewidywaniu ciągłej wartości na podstawie danych historycznych. Załóżmy, że mamy zbiór danych z poprzednimi transakcjami i wycenami pewnych akcji notowanych na giełdzie. Chcemy stworzyć algorytm, który będzie analizować dane rynkowe i przewidywać przyszłą cenę tych akcji.

Albo przypuśćmy, że jesteś menedżerem projektu. Wiesz, że oszacowania liczby godzin potrzebnej do sfinalizowania projektu zwykle nie są najlepsze: czasem Twój zespół przeszacowuje wymagany czas i kończy projekt znacznie szybciej, a czasem niedoszacowuje czas i realizacja projektu trwa znacznie dłużej, niż oczekiwano. Zarówno niedoszacowywanie, jak i przeszacowywanie przyczynia się do napięć między kierownictwem a szeregowymi pracownikami. Dotyczy to szczególnie tworzenia oprogramowania, w którym dokładne oszacowanie czasu projektu jest bardzo trudne. Nie ma wątpliwości, że niedoszacowanie liczby godzin potrzebnej do zrealizowania projektu jest niepożądane. Wydawałoby się, że z dwojga złego lepiej nieco przeszacować czas, bo jeśli projekt zostanie ukończony szybciej, to wszyscy będą zadowoleni, prawda? Niestety tak to nie działa. Kiedy powstaje harmonogram, do projektu przydziela się różne zasoby, więc jeśli projekt zostanie wykonany przedwcześnie, wpływa to na inne zaplanowane projekty. Kierownictwo musi jakoś zagospodarować nagle uwolnione zasoby. Prawdopodobnie oznacza to również, że klientowi zaproponowano zawyżoną cenę za projekt — jeśli będzie się to regularnie powtarzać, firma zacznie tracić klientów na rzecz tańszych konkurentów.

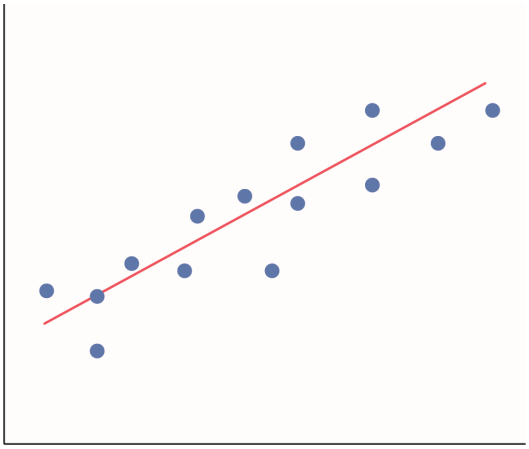
Należy zauważyć, że oszacowań często dokonują doświadczeni pracownicy, którzy robią, co w ich mocy. Szacowanie jest jednak trudne ze względu na liczbę zmiennych, które mogą wpływać na termin ukończenia projektu. Czasem w połowie projektu zmieniają się wymagania. Czasem na początku zespół nie dysponuje wszystkimi informacjami, które pojawiają się stopniowo w miarę postępów projektu. W wielu przypadkach zaimplementowanie jakiejś funkcji zależy od tego, czy strona trzecia dostarczy jakiś kluczowy komponent, a te często się opóźniają. Są to wszystko zmienne, co do których osoba oszacowująca projekt musi poczynić założenia. Oczywiście można powiedzieć, że dałoby się łatwo uniknąć wszystkich tych problemów, gdyby zespół nigdy nie zaczynał żadnego

projektu, dopóki nie będzie miał wszystkiego, czego mu potrzeba. Ale strategia ta byłaby niepraktyczna w dynamicznych środowiskach, takich jak firmy tworzące oprogramowanie albo działające w innej branży technologicznej. Innowacje niemal z definicji wiążą się z niepewnością, a niepewność oznacza, że inżynierowie często nie mogą przewidzieć wszystkich potencjalnych problemów, więc szacowanie czasu potrzebnego na rozwiązanie nienapotkanych jeszcze problemów nie jest nauką ścisłą.

Jednym z możliwych rozwiązań problemu szacowania czasu trwania projektu jest algorytm regresji. Jeśli zespół kierowniczy prowadzi bazę danych z istotnymi cechami przeszłych projektów i rzeczywistym czasem potrzebnym na ich realizację, możliwe będzie utworzenie sieci neuronowej, która przeanalizuje dane i nauczy się przewidywać czas trwania podobnych przyszłych projektów. Jak pamiętasz z poprzednich przykładów klasyfikacji, *cechy* są wymiarami danych. W przypadku analizy odczuć zdanie przekształcaliśmy w wektor liczb, a każda wartość w wektorze była cechą tego zdania. W przypadku rozpoznawania odręcznie pisanych cyfr obraz konwertowaliśmy na wektor wartości pikseli, w którym każda wartość była cechą obrazu. W przykładzie z szacowaniem czasu trwania projektu próbką danych byłby wektor cech opisujących dany projekt: liczba przydzielonych inżynierów, poziom doświadczenia inżynierów, liczba komponentów, od których zależy projekt, pora roku, w której projekt jest realizowany, itd., natomiast etykietą do przewidzenia byłby szacowany czas — powiedzmy 160 godzin. Różnica między algorytmem sieci neuronowej a ludzkim estymatorem polega na tym, że algorytm powinien być w stanie znaleźć wzorce w danych i relacje między cechami lepiej niż człowiek. Człowiek może mieć skłonność do kładzenia zbyt dużego nacisku na konkretną cechę — na przykład liczbę przydzielonych inżynierów — i szacowania czasu ukończenia pracy na podstawie liczby dostępnych inżynierów. Natomiast sieć neuronowa w procesie treningu powinna być w stanie wybrać cechy predykcyjne dla każdego indywidualnego projektu. W jednym przypadku liczba inżynierów może rzeczywiście być czynnikiem decydującym, ale w innym ważniejszy może być związek między innymi cechami. Na przykład pora roku, w której realizowany jest projekt, i firma klienta mogą być ze sobą skorelowane: w danej firmie szczyt lata może być czasem, gdy większość pracowników jest na urlopach, więc uzyskanie od nich istotnych informacji zwrotnych może być opóźnione. Te niuanse w relacjach między cechami opisującymi projekt mogą być trudne do wyłapania przez

człowieka, zwłaszcza gdy wektory projektu zawierają wiele różnych cech. Ale dla algorytmu sieci neuronowej wychwycenie takich szczegółów powinno być względnie łatwe.

Algorytmy prognostyczne tego typu nazywa się algorytmami regresyjnymi. Najprostszą odmianą algorytmu regresyjnego jest taki, który przeprowadza *regresję liniową*, którą omówię szczegółowo w rozdziale 3. Regresja liniowa to proces dopasowywania linii do zbioru punktów danych i używania jej jako estymatora przyszłych punktów. W scenariuszu regresyjnym algorytm sieci neuronowej może zautomatyzować proces znajdowania *linii najlepszego dopasowania* (zob. rysunek 1.22). Omówię teraz przykład regresji i zbudujemy estymator dla konkretnego problemu regresji. Nasz projekt recenzji filmów jest ukończony, a teraz zostaliśmy zatrudnieni przez biuro nieruchomości. Firma chce, żebyśmy zbudowali model, który pomoże szacować ceny domów w danym mieście.



Rysunek 1.22. Linia najlepszego dopasowania (*czerwona linia*) przebiegająca przez zbiór punktów danych (*niebieskie kropki*). Możemy użyć linii najlepszego dopasowania, aby przewidywać wartości brakujących próbek danych

Wiesz już, co robić. Pierwszą rzeczą, której potrzebujemy, jest zbiór próbek danych — wektorów wartości opisujących domy — wraz z poprawnymi wycenami. Istnieje zbiór danych dobrze znany inżynierom, którzy zajmują się uczeniem maszynowym. Jest to zbiór danych Boston Housing Price, który opisuje domy na przedmieściach Bostonu w latach 70. Każda z 506 próbek danych składa się z wektora o 14 wymiarach. Oznacza to, że zbiór danych obejmuje 506 domów, z których każdy jest opisany listą czternastu cech (zob. tabela 1.4).

Tabela 1.4. Opis cech w zbiorze danych Boston Housing Price

1	Wskaźnik przestępczości per capita wg dzielnicy
2	Proporcja gruntów przeznaczonych na działki o powierzchni powyżej 25 000 stóp kwadratowych pod zabudowę mieszkalną
3	Proporcja akrów powierzchni przeznaczonych pod działalność niehandlową wg dzielnicy
4	Zmienna dychotomiczna rzeki Charles (1, jeśli trakt przylega do rzeki, 0 w przeciwnym przypadku)
5	Stężenie tlenków azotu (części na 10 milionów)
6	Średnia liczba pokojów w mieszkaniu
7	Proporcja mieszkań własnościowych wybudowanych przed 1940 r.
8	Ważone odległości do pięciu bostońskich centrów zatrudnienia
9	Wskaźnik dostępności autostrad radialnych
10	Pełna stawka podatku od nieruchomości na 10 000 USD
11	Stosunek liczby nauczycieli do liczby uczniów wg dzielnicy
12	$B - 1000\,(Bk - 0{,}63)^2$, gdzie Bk to odsetek mieszkańców rasy czarnej według dzielnicy*
13	LstAt: procent populacji o niższym statusie
14	Mediana wartości domów zamieszkałych przez właścicieli (w tysiącach USD)

*__Uwaga__: *wiersz 12. wymaga wyjaśnienia; żeby jednak nie zakłócać wywodu, omówię go zaraz po zakończeniu niniejszego rozdziału*

Każdy dom w zbiorze danych jest opisany 14 wartościami odnoszącymi się do kryteriów, które mogą mieć związek z ceną domu (np. wskaźnik przestępczości per capita, liczba pokoi w domu itp.). Algorytmy uczenia maszynowego, w tym sieci neuronowe, są użytecznymi narzędziami do analizy tego typu danych dlatego, że człowiekowi bardzo trudno jest spojrzeć na każdy z 14 punktów danych w setkach domów i wybrać cechy, które najlepiej przewidują cenę domu. Intuicyjnie możemy się domyślać, że liczba pokoi jest istotna, ale jak ważna jest ona w porównaniu z lokalizacją? Albo stosunkiem liczby uczniów do nauczycieli w danej dzielnicy? Celem sieci neuronowej jest nauczenie się tych zależności na podstawie danych.

W celu rozwiązania tego problemu możemy utworzyć sieć neuronową w następujący sposób: pierwsza warstwa zawiera 14 węzłów wejściowych, po jednym na każdy wymiar danych, kolejna warstwa zawiera 64 neurony z aktywacją ReLU, a po niej następuje jeszcze jedna warstwa złożona z 64 neuronów, ostatnia warstwa składa się z jednego neuronu liniowego. Neuron liniowy to taki, który oblicza swoje wyjście na podstawie ważonej sumy wejść i wag połączeń bez żadnego dalszego przetwarzania przez funkcję aktywacji (tzn. bez modulacji progowej). Jak pamiętasz, każdy neuron w każdej warstwie jest połączony z każdym neuronem następnej warstwy. Układ sieci, jak wyjaśniłem poprzednio, dostraja się metodą prób i błędów. Pewne jest tylko to, że pierwsza warstwa powinna zawierać 14 węzłów wejściowych, a ostatnia warstwa powinna zawierać jeden neuron wyjściowy. Potrzebujemy 14 węzłów wejściowych, ponieważ nasza próbka wejściowa jest 14-wymiarowym wektorem. Podobnie potrzebujemy jednego neuronu wyjściowego, bo nasza sieć neuronowa ma przewidywać cenę domu, tzn. chcemy, aby generowała na wyjściu tylko jedną liczbę. Zamiast dwóch warstw ukrytych zawierających po 64 neurony moglibyśmy utworzyć dwie warstwy ukryte zawierające po 128 neuronów albo jedną warstwę zawierającą 128 neuronów i drugą zawierającą 64 neurony. Moglibyśmy też utworzyć trzy warstwy ukryte zawierające po 32 neurony, a we wszystkich tych przypadkach sieć neuronowa nauczyłaby się przewidywać cenę domu odzwierciedlającą, z różnymi stopniami dokładności, dane treningowe. Zwykle wypróbowalibyśmy różne układy i wybrali ten, który działa najlepiej.

Po utworzeniu sieci neuronowej proces trenowania jej na problemie regresji jest podobny do trenowania na problemie klasyfikacji. Zbiór danych treningowych, który w tym przypadku składa się z 506 domów, zawiera również docelową prognozę dla każdej próbki — innymi słowy, dla każdego wektora 14 wartości opisujących każdy dom znamy również rzeczywistą cenę tego domu. Trening jest podzielony na wiele epok (tzn. cykli, w których sieć neuronowa widzi wszystkie próbki w zbiorze danych przynajmniej raz). Na początku treningu, w pierwszej epoce, parametry sieci neuronowej są inicjalizowane losowo, przez co przewidywana cena każdego domu nie jest zbyt dokładna. Na szczęście znamy prawdziwe ceny domów, więc możemy zmierzyć, jak bardzo prognozy sieci różnią się od prawdziwej wartości domu. Korzystając z tych informacji (i wielu obliczeń), aktualizujemy parametry sieci neuronowej, aby następnym razem zminimalizować różnicę między wartością oczekiwaną a przewidywaną. Tak jak

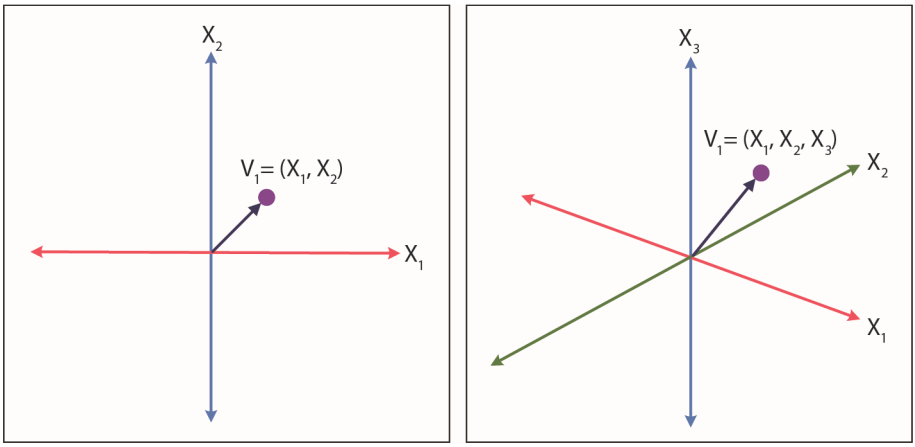
w przykładzie klasyfikacji, na przestrzeni wielu epok parametry sieci neuronowej (wagi połączeń) są stopniowo aktualizowane i dostrajane w celu dokładnego przewidywania wartości domu na podstawie jego deskryptorów (cech). Liczymy na to, że w trakcie treningu sieć neuronowa nauczy się związku między cechami opisującymi dom a wartością domu. Następnie możemy zaprezentować jej nowy dom, który nie należy do zbioru danych treningowych, a sieć neuronowa powinna być w stanie przewidzieć jego wartość.

WEKTORY I PRZESTRZENIE WEKTOROWE

Poświęciliśmy sporo czasu na zrozumienie w pełni połączonych sieci neuronowych oraz ich użycia do klasyfikacji i regresji, dwóch najczęstszych typów problemów w dziedzinie sztucznej inteligencji. Wiesz już, jak warstwa po warstwie zbudować sieć neuronów, która przyjmuje wektor wejściowy i generuje albo pojedynczą wartość, albo jakiś wektor predykcyjny. Wspomniałem też mimochodem, że algorytm klasyfikacji (w tym sieć neuronową) można opisać jako linię, która rozdziela punkty w przestrzeni *N*-wymiarowej. Punkty po jednej stronie linii należą do jednej klasy, a punkty po drugiej stronie linii należą do drugiej klasy. Podobnie stwierdziliśmy, że regresyjną sieć neuronową można traktować jak linię najlepszego dopasowania przechodzącą przez zbiór punktów danych, przy czym sama linia jest ciągłym predyktorem dla wszystkich punktów danych w tej *N*-wymiarowej przestrzeni. Problem z tym wyjaśnieniem polega na tym, że trudno je zwizualizować.

Przypuśćmy, że chcemy odróżniać obrazy napisanej odręcznie cyfry 1 od napisanej odręcznie cyfry 2. Analogia „linii rozdzielającej punkty" z pewnością nie ma tu zastosowania, prawda? Jak można myśleć o obrazach jako o punktach i jak użyć linii do rozdzielenia obrazów? Okazuje się, że istotnie można traktować obrazy — albo dowolne inne próbki danych, w tym domy i recenzje filmowe — jak punkty w pewnej przestrzeni. Przestrzeń ta jest zwykle wielowymiarowa, więc nazywamy ją *hiperprzestrzenią*, a sieć neuronowa naprawdę próbuje znaleźć linię, która rozdziela te punkty danych; ponieważ jednak mamy do czynienia z wieloma wymiarami, zamiast linii (która rozdziela punkty w dwóch wymiarach), próbujmy znaleźć *hiperpłaszczyznę*. Aby zrozumieć ten proces, cofnijmy się do czasów szkoły średniej i przypomnijmy sobie, czym jest wektor.

W przestrzeni dwuwymiarowej (2D) wektor można opisać następująco: $V = (X_1, X_2)$, gdzie X_1 opisuje komponent wektora w jednym wymiarze, a X_2 opisuje komponent wektora w drugim wymiarze. Wektor w przestrzeni trójwymiarowej (3D) można opisać następująco: $V = (X_1, X_2, X_3)$, gdzie X_1, X_2 i X_3 opisują komponenty wektora w każdym z trzech wymiarów. Możemy zamknąć oczy i wyobrazić sobie punkt w przestrzeni 2D ze strzałką zaczynającą się w środku układu współrzędnych i kończącą w tym punkcie. Jest to wektor dwuwymiarowy. Możemy też wyobrazić sobie wektor w przestrzeni 3D jako punkt pływający gdzieś w trójwymiarowym świecie ze strzałką zaczynającą się w środku układu współrzędnych i kończącą w tym punkcie (zob. rysunek 1.23). Czy potrafisz jednak zwizualizować sobie wektor w przestrzeni 4D? Niestety nie umiemy wizualizować sobie relacji przestrzennych wykraczających poza trzy wymiary. Okazuje się jednak, że matematyka wektorów w czterech wymiarach jest taka sama jak matematyka wektorów w dwóch i trzech wymiarach. Choć nie umiemy go już sobie wyobrazić, wektor 4D matematycznie wygląda tak: $V = (X_1, X_2, X_3, X_4)$, zupełnie jak wektor 3D, ale z jednym dodatkowym wymiarem. Natomiast wektor 5D wygląda tak: $V = (X_1, X_2, X_3, X_4, X_5)$, a wektor *N*-wymiarowy wygląda tak: $V = (X_1, X_2, X_3, \ldots, X_N)$. Zatem manipulowanie wektorami w przestrzeni wielowymiarowej przypomina manipulowanie wektorami w przestrzeniach 2D i 3D, które są dla nas bardziej intuicyjne. Musimy po prostu uwzględnić dodatkowe wymiary.



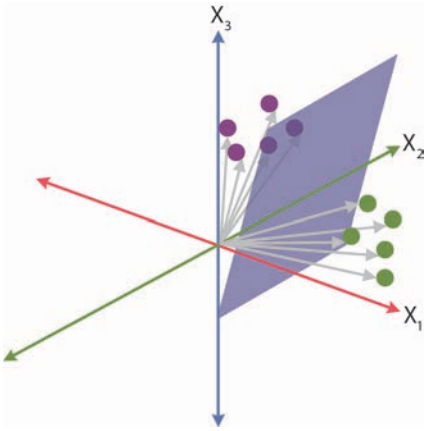
Rysunek 1.23. Wektor w przestrzeni 2D (po *lewej* stronie) i wektor w przestrzeni 3D (po *prawej* stronie). Wektor to po prostu punkt w przestrzeni ze strzałką biegnącą od początku układu współrzędnych do tego punktu

Do czego może się to przydać? Wróćmy do obrazu napisanej odręcznie cyfry 1 o rozdzielczości 28×28 pikseli. Aby przetworzyć ten obraz w naszej sieci neuronowej, zaczęliśmy od przekształcenia go w wektor. Zinterpretowaliśmy kolor każdego piksela jako liczbę, a następnie utworzyliśmy wektor złożony z 784 ($28 \cdot 28$) liczb. Oznacza to, że utworzyliśmy wektor $V = (X_1, X_2, X_3, \ldots, X_{784})$ mający 784 wymiary. Zdarzyło się tu coś bardzo subtelnego i niezwykle ważnego. Koncepcyjnie przeszliśmy od interpretowania obrazu jako zbioru 784 pikseli o niezależnych kolorach do interpretowania go jako pojedynczego punktu w 784-wymiarowej przestrzeni. Jest to bardzo istotne założenie. Stwierdzamy, że 784 piksele nie są po prostu losowymi wartościami, które przypadkiem przypominają obraz cyfry 1, ale że obraz ten istnieje jako pojedynczy punkt w 784-wymiarowym wszechświecie, w którym każda wartość piksela jest komponentem wektora w jednym z 784 wymiarów. Jeśli rozważymy inne próbki pisanej odręcznie cyfry 1, wszystkie one również będą pojedynczymi punktami w wirtualnym 784-wymiarowym wszechświecie.

A teraz pora na coś naprawdę interesującego. Wektory mają pewną szczególną właściwość. Wiemy, że podobne wektory wskazują w przybliżeniu ten sam kierunek, a wektory, które są różne, wskazują różne kierunki. Wykorzystując intuicję nabytą w przestrzeniach 2D i 3D, moglibyśmy sobie wyobrazić, że w świecie o 784 wymiarach również istnieje punkt początkowy, a z punktu tego wychodzą strzałki do każdego z punktów reprezentujących odręcznie pisane jedynki. Teraz, żeby było ciekawiej, dorzućmy do tego obrazy piątek. One też mają 28×28 pikseli, więc istnieją w tym samym 784-wymiarowym wszechświecie co obrazy jedynek, ale ponieważ są to inne cyfry, co to może oznaczać w kategoriach przestrzeni wektorowej? Jeśli odpowiedziałeś, że wektory piątek będą wskazywały inne kierunki niż wektory jedynek, to masz rację. Zasługujesz na przerwę — idź i napij się czegoś zimnego! Jeśli odpowiedziałeś inaczej, może wróć na początek tego podrozdziału i przeczytaj go jeszcze raz. Jest to ważna koncepcja, która ma kluczowe znaczenie dla uczenia maszynowego i sztucznej inteligencji.

Interpretowanie próbek danych jako wektorów w jakiejś *N*-wymiarowej przestrzeni oznacza, że można sobie wyobrazić próbki tej samej klasy zgrupowane bliżej siebie w tej przestrzeni, a próbki innej klasy zgrupowane w nieco innym miejscu. Teraz, gdy mamy sposób interpretowania danych jako grup punktów, algorytm klasyfikacji musi po prostu znaleźć hiperpłaszczyznę (pamiętaj: linię w dwóch wymiarach, płaszczyznę w trzech wymiarach i hiperpłaszczyznę w wielu wymiarach), która przecina przestrzeń wektorową na pół w taki sposób,

że punkty po jednej stronie płaszczyzny należą do jednej klasy, a punkty po drugiej stronie należą do innej klasy (zob. rysunek 1.24). Dlatego wektory są tak przydatne. Pomagają interpretować dane w sposób, który pozwala zastosować koncepcje z algebry i geometrii w celu odkrycia istotnych relacji między próbkami danych.



Rysunek 1.24. Dwie klasy wektorów w przestrzeni 3D rozdzielone hiperpłaszczyzną. Wektory po lewej stronie płaszczyzny (wskazujące *fioletowe punkty*) należą do jednej klasy, a wektory po prawej stronie płaszczyzny (wskazujące *zielone punkty*) należą do drugiej klasy. Dlaczego jest to użyteczne? Jeśli otrzymamy nową próbkę i nie będziemy wiedzieli, do jakiej należy ona klasy, wystarczy, że zinterpretujemy ją jako wektor i sprawdzimy, po której stronie płaszczyzny się znajduje — w ten sposób będziemy mogli przewidzieć jej klasę

○ ● ○

W tym rozdziale naszym celem było zrozumieć podstawy działania sztucznych sieci neuronowych. Trudno nie przypisywać tym narzędziom analitycznym mistycznych właściwości, zważywszy na ich nazwę. Jak jednak zaczynasz rozumieć, są to po prostu konstrukcje matematyczne złożone z prostych operacji, głównie mnożenia i dodawania, zorganizowanych w sekwencję kroków (krokami tymi oczywiście są warstwy neuronów ułożone jedna za drugą). Rozdział zaczęliśmy od omówienia historii sztucznych sieci neuronowych i pierwszego sztucznego neuronu, neuronu McCullocha-Pittsa. Dowiedziałeś się, że inspiracją do powstania sztucznego neuronu był ludzki mózg oraz sposób, w jaki neurony biologiczne przetwarzają sygnały wejściowe. Neuron McCullocha-Pittsa, jako pierwsza inkarnacja sztucznego neuronu, był prostą konstrukcją. Spełniał minimum wymagań koniecznych do tego, żeby w ogóle można było nazwać go neuronem.

W miarę postępów w rozumieniu systemów biologicznych, dzięki badaniom naukowców takich jak Donald Hebb, dowiedzieliśmy się, że nie wszystkie sygnały wejściowe są traktowane jednakowo. Istnieje system wag, który można regulować, aby nadać większe znaczenie sygnałom pochodzącym z niektórych wejść. Badania biologiczne przeniesiono na grunt informatyki (choć wówczas nauka ta nie była jeszcze znana jako informatyka), a do sztucznego neuronu dodano wagi. Postępy te doprowadziły nas do perceptronu Rosenblatta i pierwszych sieci neuronowych, które potrafiły robić coś użytecznego, ADALINE i MADALINE.

Dowiedziałeś się również, że pierwsze architektury sieci neuronowych — mianowicie perceptrony jednowarstwowe — miały istotną wadę: można było używać ich wyłącznie do klasyfikowania punktów danych rozdzielnych liniowo. Wiesz już, co to oznacza, ponieważ omówiłem analizę próbek danych (obrazów, tekstu) w kategoriach punktów (wektorów) w przestrzeni wielowymiarowej. Zbiór danych rozdzielny liniowo to po prostu taki, którego klasy można rozdzielić na dwie części za pomocą linii lub hiperpłaszczyzny. Niestety w praktyce większość problemów klasyfikacyjnych nie jest rozdzielna liniowo w swojej naturalnej przestrzeni. Na przykład, gdybyśmy chcieli klasyfikować obrazy kotów i psów i użylibyśmy klasyfikatora liniowego bezpośrednio na zdjęciach kotów i psów, okazałoby się, że działa on słabo. „Surowe" wektory — innymi słowy wektory utworzone w wyniku przekształcenia obrazów w tablice wartości pikseli — to wektory, które zwykle zawierają wiele danych nieistotnych dla problemu (np. piksele tła). Biorąc pod uwagę te dodatkowe piksele i szum w naturalnych obrazach, trudno oczekiwać, że jako wektory będą one tworzyć uporządkowaną przestrzeń z kotami po jednej stronie, psami po drugiej i hiperpłaszczyzną pośrodku. Sieci neuronowe z dodatkowymi warstwami (czyli perceptrony *wielowarstwowe*) pomagają przekształcić wektory wejściowe w inną hiperprzestrzeń, w której wynikowe wektory *są* rozdzielne liniowo. Oto sekret sieci neuronowych! Uczą się one przekształcać wektory wejściowe w hiperprzestrzeń — innymi słowy w przestrzeń o innej liczbie wymiarów — w której wektory da się rozdzielić liniowo.

W następnym rozdziale przyjrzymy się widzeniu komputerowemu, jego historii i podobieństwu do widzenia biologicznego. Przedstawię też architekturę sieci neuronowej znaną jako *konwolucyjna sieć neuronowa* (ang. *convolutional neural network*, CNN), która stała się standardem w systemach widzenia komputerowego.

Zanim jednak przejdziemy do następnego rozdziału, muszę wyjaśnić ważną kwestię.

PUNKT PRZERWANIA: KONFRONTACJA Z PROBLEMATYCZNYM DZIEDZICTWEM

Dwunasty wiersz zbioru danych Boston Housing Price określa „odsetek mieszkańców rasy czarnej według dzielnicy". Zbiór ten opracowano w latach 70., a cecha ta (wymiar próbki danych) jasno ilustruje ówczesne uprzedzenia. Dlaczego taki zbiór danych jest przykładem w niniejszej książce? Jest to jeden z wielu „zabawkowych" zbiorów danych udostępnionych w internecie na potrzeby badań związanych z uczeniem maszynowym i sztuczną inteligencją. Od samouczków i książek poświęconych uczeniu głębokiemu do konkursów online, zbiór danych Boston Housing Price jest często używany do oceniania modeli AI i porównywania trafności nowych modeli z bieżącym stanem techniki. Choć jednak zbiór ten jest powszechnie znany w społeczności AI, niemal nikt nie wspomina o jego 12. wierszu.

Można by podać wiele możliwych przyczyn takiego stanu rzeczy, ale podejrzewam, że odpowiedź jest prosta i powinna budzić poważny niepokój. Otóż większość ludzi nie przygląda się danym wystarczająco uważnie, żeby w ogóle to zauważyć. Problem z 12. wierszem powinien być oczywisty. Sugeruje on, że istnieje zależność między liczbą czarnoskórych osób w dzielnicy a wartością domów. Odpowiedź apologety jest równie oczywista: co jest złego w umieszczeniu tej informacji w systemie? Jeśli okaże się, że model znajdzie zależność między czarnoskórymi osobami a ceną domów, chyba nie jest to moja wina? Z taką interpretacją wiąże się wiele problemów.

Zajrzyj jeszcze raz do tabeli 1.4 i przejrzyj wszystkie 14 cech zbioru danych. Za każdym razem gdy tworzymy zbiór danych treningowych, istnieje ryzyko, że zbiór ten będzie *stronniczy*. Rzekłbym wręcz, że zbiór danych treningowych jest zawsze w jakiś sposób stronniczy, a wady tej niemal nie sposób uniknąć. Zbiór danych konstruuje się poprzez wybieranie próbek, które zostaną użyte do trenowania modelu. To, że wybiera się jedne próbki, a nie inne, wprowadza do systemu tendencyjność. Pomyśl o zbiorze pisanych odręcznie cyfr. Zawiera on 50 000 próbek treningowych. Nie wiadomo, ile osób pisało cyfry uwzględnione w tym zbiorze danych, ale załóżmy, że 5000 osób napisało po 10 cyfr. W przypadku 300-milionowej populacji nie jest jasne, czy pismo 5000 wybranych osób dobrze reprezentuje sposób, w jaki większość ludzi pisze cyfry. Jeśli zatem zbiór próbek zawiera cyfry napisane przez zaledwie 5000 osób, możliwe, że trenujemy

model z pewnym ukierunkowaniem na sposób pisania tych 5000 osób, co może źle się przekładać się na ogólną populację. Oczywiście poziom zagrożeń związanych ze stronniczością bywa różny. Jeśli zbiór pisanych odręcznie cyfr rzeczywiście jest ukierunkowany na to, jak kilka osób pisze cyfry, najgorsze, co może się zdarzyć, to to, że model nie będzie dobrze generalizował się w środowisku produkcyjnym i będzie popełniał błędy w interpretowaniu odręcznego pisma ogółu populacji.

Wróćmy teraz do tabeli 1.4. Od razu widać, że ten zbiór cechuje się większą stronniczością, nawet jeśli pominiemy wiersz 12. Zbiór danych składa się z 14 cech służących do przewidywania ceny domu. Przez sam akt wyboru położyliśmy większy nacisk na te 14 cech niż na jakikolwiek inny predyktor cen domów. Teraz rozważ wiersz 12. Kiedy dołączamy liczbę czarnoskórych jako możliwy predyktor cen domów, sprawiamy, że zbiór danych staje się stronniczy w stosunku do czarnoskórych osób — bez względu na to, czy dane pokazują pozytywną, czy negatywną korelację między cechą a ceną domu. Jednak w przeciwieństwie do przykładu z odręcznie pisanymi cyframi ta tendencyjność ma fatalne konsekwencje. Wiersz 12. może wpływać na dane na trzy różne sposoby: czarnoskórzy przyczyniają się do podwyższenia cen domów, czarnoskórzy przyczyniają się do obniżenia cen domów, czarnoskórzy nie są predyktorem cen domów. Tylko jeden z trzech możliwych wyników jest pozytywny dla czarnoskórych. Od razu oznacza to mniejszą szansę, że zbiór ten będzie korzystny dla czarnoskórych, a fakt, że zostali oni wybrani jako cecha danych, niesprawiedliwie stawia ich w pozycji podsądnego w symulowanym procesie.

Przypuśćmy, że nasz model rzeczywiście znajdzie związek między liczbą czarnoskórych osób a cenami domów, i przypuśćmy, że będzie to związek negatywny. Tym, czego model nie może wyjaśnić, jest przyczyna tego związku. Czy wynik oznacza, że jeśli czarnoskórzy sprowadzą się do zamożnej dzielnicy, ceny nieruchomości spadną? Czy może oznacza, że z powodu dobrze udokumentowanej „rasowej luki bogactwa”[7] czarne osoby są średnio uboższe i mogą sobie pozwolić tylko na mieszkanie w okolicach, w których ceny nieruchomości są już niskie? Społeczne uprzedzenia analityka mogą łatwo wypaczyć jego interpretację

[7] V. Williamson, *Closing the Racial Wealth Gap Requires Heavy, Progressive Taxation of Wealth*, Brookings Institution, 9 grudnia 2020 [dostęp: 22.07.2024], *https://www.brookings.edu/research/closing-the-racial-wealth-gap-requires-heavy-progressive-taxation-of-wealth/*.

wyników i doprowadzić go do wniosku, że „czarnoskórzy źle wpływają na wartość nieruchomości w okolicy". Taka interpretacja może prowadzić do polityki segregacji: mieszkańcy mogą zdecydować, że nie pozwolą czarnym na osiedlanie się w dzielnicy, żeby wartość ich nieruchomości nie spadła. To prowadzi do dalszej alienacji czarnoskórych i ograniczania ich praw obywatelskich.

Drugim problemem z tym przykładem (jeśli pierwszy nie był wystarczająco niepokojący) jest statystyczna istotność danych. Zbiór danych zawiera 506 przykładowych domów. Czy wiemy, jak rozkładają się te próbki na terenie Bostonu? Czy bez tej wiedzy możemy wyciągać wnioski na temat związków między danymi wejściowymi a wynikami modelu? A jednak łatwo byłoby wykorzystać uzyskane wyniki do wspierania stronniczej polityki. Załóżmy, że w tym zbiorze danych istnieje ujemna zależność między czarnoskórymi a cenami domów. A co z innymi zbiorami danych? Co by było, gdyby zbiór danych był większy i obejmował inne rejony, w których średnie dochody czarnych gospodarstw domowych są wyższe? A co by było, gdyby zbiór danych obejmował inne grupy etniczne: czy wiemy, jak wpływają Irlandczycy lub Włosi na wartość nieruchomości? A co z Żydami? Wszyscy wiemy, jak to się kończy.

Ten zbiór danych powstał w latach 70. XX wieku i z definicji jest rasistowski. Jednak wiele osób używa go współcześnie, nie zdając sobie sprawy z jego rasistowskiego aspektu, co jest istotnym zagrożeniem, o którym trzeba tu wspomnieć. Mamy dziś dostęp do większej ilości danych, niż ludzkość miała w całej historii naszej cywilizacji. Wydaje się też, że po raz pierwszy mamy możliwość modyfikowania danych, przekształcania ich oraz wydobywania z nich wzorców i informacji. Nie jest jednak jasne, czy potrafimy zrozumieć wyniki i nadać im sens. Skąd wiem, że ludzie korzystają z danych, nie zwracając na nie wystarczającej uwagi? Zbiór danych dotyczących cen mieszkań w Bostonie został wykorzystany, ale nie wyjaśniony, w kilku samouczkach internetowych i co najmniej jednej książce na temat uczenia głębokiego. W większości przypadków autorzy wymieniają tylko kilka pierwszych wierszy (wiersze 1. – 3.) i wyjaśniają znaczenie tych cech, nie omawiając pozostałych wierszy. Zamiast tego pokazują po prostu przykłady wektorów 14 wartości. Nie przywiązują też większej wagi do znaczenia tych wartości.

Czy zrobiono to celowo, aby uniknąć wstydliwej części naszej historii? Twierdząca odpowiedź na to pytanie byłaby zła, ale przypuszczam, że jest przecząca, co niestety nie znaczy, iż dużo lepsza. Pokazuje to, że jesteśmy gotowi używać danych bez ich prawdziwego zrozumienia i oczekujemy, iż nasze modele

sztucznej inteligencji będą na tyle „inteligentne", aby uniknąć stronniczości w danych. Jak jednak widać w tym rozdziale, zbyt skwapliwie przypisujemy inteligencję sztucznej inteligencji. Nasze modele po prostu uczą się wzorców z danych. Jeśli wprowadzimy do nich stronnicze dane, to otrzymamy stronnicze wyniki — a nasze dane zawsze są stronnicze!

Kiedy nie rozumiemy danych i decydujemy, że musimy wykorzystać model AI, żeby nadać im sens, narażamy się na realne niebezpieczeństwo. Jeśli na ślepo wrzucimy informacje do modelu i wprowadzimy politykę na podstawie otrzymanych wyników, będziemy w pełni odpowiedzialni za utrwalanie uprzedzeń. Odpowiedzialne podejście polega na zrozumieniu implikacji naszych danych, zwłaszcza ukrytej w nich stronniczości, a następnie wykorzystaniu algorytmów do znalezienia wzorców, których my, ludzie, nie potrafimy dostrzec.

Sztuczna inteligencja jest dziś wszędzie. Sugeruje, co warto kupić, obejrzeć lub przeczytać. Wkrótce codziennością mogą się stać autonomiczne samochody, chipy wszczepiane w ludzkie ciała czy zastosowanie AI w medycynie i wymiarze sprawiedliwości. Te innowacje mogą przynieść wiele korzyści, jednak ważne jest, aby pamiętać o realnych zagrożeniach związanych z kontrolą nad technologią i jej etycznym użyciem. Mimo to większość z nas niewiele wie o tym, czym jest i jak działa AI.

Książka ta pojawia się w kluczowym momencie, pozwalając szerszej publiczności zrozumieć „czarną skrzynkę", jaką jest AI!

Dr Saumil Patel, Argonne National Lab

Lektura tej książki pozwoli Ci zrozumieć możliwości najnowocześniejszych algorytmów AI, nawet jeśli nie posiadasz wiedzy technicznej. Dowiesz się, na czym polega funkcjonowanie sieci neuronowych, poznasz również fascynującą historię pierwszego sztucznego neuronu i przekształcenia go w sieć zdolną do realizowania zadań wcześniej uznanych za niewykonalne obliczeniowo. Zapoznasz się z możliwościami i wyzwaniami związanymi z widzeniem komputerowym, a także z zasadami działania sieci neuronowej i jej treningu. Zorientujesz się też, w których obszarach sztuczna inteligencja może stanowić zagrożenie, a gdzie wykazuje olbrzymi potencjał. Na koniec spojrzysz na obecną rewolucję AI przez pryzmat wcześniejszych przełomów technologicznych, aby lepiej zrozumieć, co nas czeka i jakie mamy wobec tego zobowiązania.

Fascynujący przewodnik autostopowicza po galaktyce AI!

Dr Alireza Sadeghian,
Artificial Intelligence Lab, Toronto Metropolitan University

Kenneth Wenger zajmuje się sztuczną inteligencją w kontekście determinizmu, pracując nad zapewnieniem bezpiecznego działania sieci neuronowych w systemach krytycznych. Oprócz badań interesuje się ludźmi i tym, jak technologia wpływa na społeczeństwo. Mieszka z rodziną w Mississauga w prowincji Ontario.

Helion
helion.pl
HELION S.A.
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl
KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-289-1636-4
9 788328 916364
Cena: 59,00 zł