Ian Goodfellow ▪ Yoshua Bengio ▪ Aaron Courville

DEEP LEARNING

Systemy uczące się

PWN

DEEP LEARNING

Ian Goodfellow ▪ Yoshua Bengio ▪ Aaron Courville

DEEP LEARNING

Systemy uczące się

PWN

Przekład **WITKOM Witold Sikorski**

Projekt okładki polskiego wydania **Hubert Zacharski**

Ilustracja na okładce: Central Park Azalea Walk Dreamscape, autor – Daniel Ambrosi (danielambrosi.com) Dreamscapes Daniela Ambrosiego są tworzone dzięki zastosowaniu otwartego oprogramowania DeepDream Google'a, zmodyfikowanego przez Josepha Smarra (Google) i Chrisa Lamba (NVIDIA) tak, aby z powodzeniem działało na panoramicznych obrazach złożonych z wielu setek pikseli tworzonych przez Ambrosiego.

Wydawca **Łukasz Łopuszański**

Redaktor prowadzący **Adam Kowalski**

Redaktor **Joanna Forysiak**

Koordynator produkcji **Anna Bączkowska**

Skład i łamanie **Fixpoint**

ISBN: 978-83-01-19583-0

Wydanie I
Warszawa 2018

Wydawnictwo Naukowe PWN SA
tel. 22 69 54 321; faks 22 69 54 288
infolinia 801 33 33 88
e-mail: pwn@pwn.com.pl, reklama@pwn.pl
www.pwn.pl

Druk i oprawa: OSDW Azymut Sp. z o.o.

Spis treści

1

Wprowadzenie

Wynalazcy od dawna marzyli o myślących maszynach. Te marzenia sięgają przynajmniej do czasów starożytnej Grecji. Mityczne postaci Pigmaliona, Dedala czy Hefajstosa można interpretować jako legendarnych wynalazców, a Galateę, Talosa i Pandorę rozpatrywać jako sztuczne życie (Ovid and Martin 2004, Sparkes 1996, Tandy 1997).

Już wtedy, gdy powstały pierwsze programowane komputery, ludzie zaczęli się zastanawiać, czy mogą one stać się inteligentne (Lovelace 1842) – ponad 100 lat wcześniej zanim zbudowano pierwsze inteligentne maszyny. Dziś **sztuczna inteligencja** (AI) to ekscytująca dziedzina z wieloma zastosowaniami praktycznymi i programami badawczymi. Wprowadza się inteligentne oprogramowanie, by zautomatyzować rutynowe prace, zrozumieć mowę lub obrazy, wykonywać diagnostykę medyczną i wspomagać podstawowe badania naukowe.

We wczesnych latach sztucznej inteligencji szybko rozwiązywane były problemy trudne do wykonania dla ludzi, ale dość proste dla komputerów – problemy, które można opisać jako listy sformalizowanych matematycznych reguł. Prawdziwym wyzwaniem dla sztucznej inteligencji okazało się rozwiązywanie zadań łatwych do wykonania dla ludzi, ale trudnych w opisie formalnym – problemów, które rozwiązujemy intuicyjnie, niejako automatycznie, jak rozpoznawanie słów lub twarzy na obrazach.

Książka ta poświęcona jest rozwiązywaniu właśnie tych intuicyjnych przeszkód. By je pokonać, trzeba pozwolić komputerom, aby uczyły się na swoich doświadczeniach i rozumiały świat w kontekście hierarchii pojęć, z których każde jest zdefiniowane przez jego związek z pojęciem prostszym. Dzięki gromadzeniu wiedzy na podstawie doświadczeń unika się potrzeby formalnego opisania przez ludzi całej wiedzy potrzebnej komputerowi. Hierarchia pojęć

pozwala mu na uczenie się skomplikowanych pojęć przez budowanie ich z pojęć prostszych. Jeśli narysujemy schemat, w którym pojęcia te nakładają się na siebie, to powstanie wiele warstw. Dlatego to podejście do AI nazywamy *głębokim uczeniem się*[*].

Wiele początkowych sukcesów na polu AI miało miejsce we względnie sterylnych i sformalizowanych środowiskach i nie wymagało od komputerów specjalnej wiedzy o świecie. Na przykład system gry w szachy komputera Deep Blue firmy IBM pokonał w 1997 roku mistrza świata Garriego Kasparowa (Hsu 2002). Szachy to prosty świat, obejmujący tylko 64 pola i 32 elementy, które mogą poruszać się według ściśle określonych reguł. Opracowanie prowadzącej do sukcesu strategii gry w szachy to niezwykłe osiągnięcie, ale nie wynika ono z trudności opisania komputerowi figur szachowych i dopuszczalnych ruchów. Szachy można dokładnie opisać za pomocą krótkiej listy całkiem sformalizowanych reguł podanych przez programistę.

Jak na ironię, abstrakcyjne i sformalizowane zadania, należące do najtrudniejszych dla człowieka, są najłatwiejszymi dla komputera. Komputery od dawna mogą pokonać najlepszego szachistę, ale dopiero ostatnio zaczęły nabywać niektóre zwykłe ludzkie umiejętności, jak rozpoznawanie obiektów czy mowy. Życie codzienne wymaga sporej wiedzy o świecie. Jej duża część jest subiektywna i intuicyjna, dlatego trudno ją formalnie opisać. Aby działać inteligentnie, komputery muszą zdobyć tę samą wiedzę. Jednym z kluczowych wyzwań w dziedzinie sztucznej inteligencji jest zatem przekazanie tej nieformalnej wiedzy komputerowi.

W kilku projektach z dziedziny sztucznej inteligencji poszukiwano twardej wiedzy o świecie podanej językiem sformalizowanym – komputer może automatycznie tworzyć argumenty na podstawie tych instrukcji za pomocą logicznych reguł wnioskowania. Nazywa się to podejściem do sztucznej inteligencji **opartym na wiedzy**. Żaden z tych projektów nie zakończył się sukcesem. Najbardziej znany jest Cyc (Lenat and Guha 1989). Cyc to mechanizm wnioskowania i baza danych instrukcji w języku o nazwie CycL. Instrukcje te są wprowadzane przez personel złożony z ludzi nadzorujących projekt. Jest to uciążliwy proces. Ludzie męczą się, tworząc sformalizowane reguły o takiej złożoności, aby opisywały świat. Cyc nie potrafił zrozumieć opowieści o człowieku imieniem Fred, który rano się golił (Linde 1992). Jego mechanizm rozumowania wykrył niespójność w tej opowieści: wiedział,

[*]Termin *machine learning* jest tłumaczony na język polski na kilka sposobów. W tłumaczeniu przyjęto określenie *systemy uczące się*, aby podkreślić jednoznacznie, że systemy same się uczą, a nie uczą kogoś, co sugeruje sformułowanie *uczenie maszynowe*. Podobnie więc mamy *głębokie uczenie się*. W tekście zachowano jednak określenie *deep learning*, jako nowe pojęcie, którego przekład nie jest w Polsce utrwalony (wszystkie przypisy oznaczone * pochodzą od tłumacza).

że ludzie nie mają części elektrycznych, ale Fred miał w ręku elektryczną golarkę, więc uwierzył, że jednostka „FredPodczasGolenia" zawierała elementy elektryczne. Stąd pojawiło się pytanie, czy podczas golenia Fred nadal jest osobą.

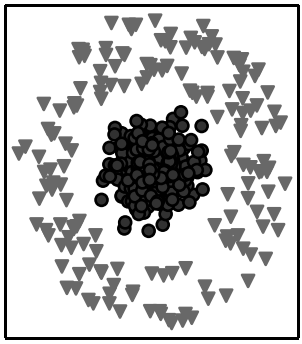
Trudności systemu opierającego się na twardo zapisanej wiedzy sugerują, że systemy AI potrzebują zdolności do przyswajania własnej wiedzy wyodrębnianej jako wzorce z surowych danych. Takie zdolności określa się mianem **systemów uczących się**. Ich wprowadzenie pozwoliło komputerom na zajęcie się problemami obejmującymi wiedzę o świecie i podejmowanie decyzji, które wydają się subiektywne. Prosty algorytm samouczenia się zwany regresją logistyczną pozwala określić, czy zalecać cesarskie cięcie (Mor-Yosef et al. 1990). Inny prosty algorytm samouczenia zwany „naiwny Baynes" potrafi oddzielić ważne e-maile od spamu.

Działanie tych algorytmów mocno zależy od reprezentatywności otrzymanych danych. Na przykład, gdy regresja logistyczna jest używana do wskazania cesarskiego cięcia, system AI nie bada pacjenta. To lekarz podaje systemowi kilka odpowiednich informacji, jak obecność blizn macicznych lub ich brak. Każdy fragment informacji ujęty w opisie pacjentki jest określany jako cecha. Regresja logistyczna uczy się, jaki związek ma każda z cech pacjentki z rozmaitymi rezultatami. Nie może jednak w żaden sposób wpływać na sposób definiowania tych cech. Jeśli regresja logistyczna otrzyma skan rezonansu magnetycznego (MRI) pacjentki zamiast formalnego raportu lekarza, nie potrafi dać użytecznych wskazań. Pojedyncze piksele skanowania MRI mają pomijalną korelację z jakimikolwiek komplikacjami, jakie mogą pojawić się przy porodzie.

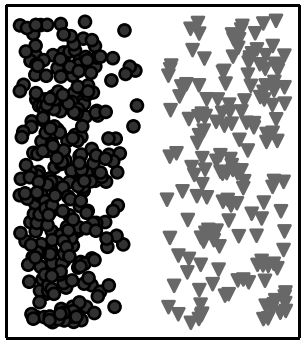
Ta zależność od sposobu reprezentacji jest ogólnym zjawiskiem w całej dziedzinie informatyki, a także w życiu codziennym. W naukach komputerowych działania takie, jak przeszukiwanie zbioru danych, są w wykładniczo szybsze, jeśli zbiór ma strukturę i jest inteligentnie indeksowany. Ludzie z łatwością wykonują działania arytmetyczne na cyfrach arabskich, ale działania na cyfrach rzymskich zajmują im znacznie więcej czasu. Nie jest więc zaskoczeniem, że wybrany sposób reprezentacji ma ogromny wpływ na wydajność algorytmów samouczących się. Prosty obrazowy przykład pokazano na rysunku 1.1.

Wiele zadań z zakresu sztucznej inteligencji można rozwiązać, projektując odpowiedni dla danego zadania zestaw cech, a następnie dostarczając te cechy prostemu algorytmowi uczącemu się. Na przykład użyteczną cechą identyfikacji mówcy na podstawie dźwięku jest ocena rozmiaru jego toru głosowego. Cecha ta daje nam możne przesłanki do określenia, czy mówiący jest kobietą, mężczyzną czy dzieckiem. Jednak dla wielu zadań trudno określić

Współrzędne kartezjańskie



Współrzędne



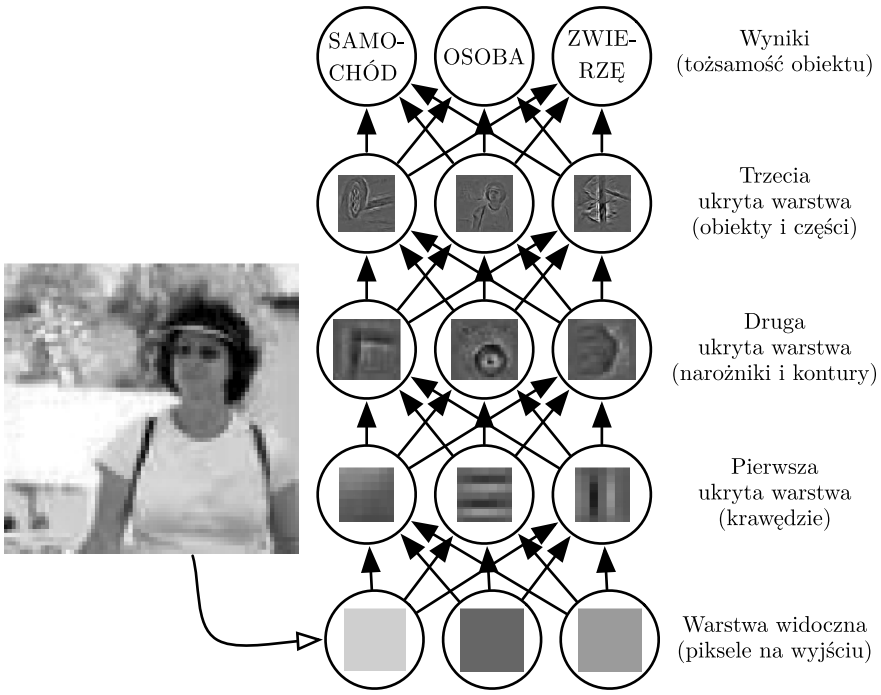
uczą się przechowywać możliwie wiele informacji, gdy dane przechodzą przez proces kodowania, a potem przez dekoder, ale także przypisują nowej reprezentacji różne właściwości. Różne rodzaje autokodowania mają na celu uzyskanie różnych rodzajów właściwości.

Gdy projektujemy cechy lub algorytmy do uczenia się cech, naszym celem jest zwykle oddzielenie czynników zmiennych, które tłumaczą obserwowane dane. W tym kontekście używamy słowa „czynniki", aby odnieść się do oddzielnych źródeł wpływu; czynniki nie są zwykle powiązane z mnożeniem. Takie czynniki często nie są bezpośrednio obserwowanymi wielkościami. Mogą występować jako nieobserwowane obiekty lub siły w świecie rzeczywistym, które wpływają na obserwowalne wielkości. Mogą także występować jako konstrukcje w umyśle człowieka, które dostarczają upraszczających wyjaśnień lub wnioskowanych przyczyn obserwowanych danych. Można je traktować jak pojęcia lub abstrakcje pomagające nam zrozumieć bogatą zmienność danych. Podczas analizy nagrań mowy czynniki zmienne obejmują wiek mówcy, jego płeć i akcent oraz wypowiadane słowa. Podczas analizy obrazu samochodu: położenie samochodu, jego kolor oraz kąt padania promieni słonecznych.

Podstawowym źródłem trudności w aplikacjach sztucznej inteligencji jest fakt, że wiele czynników zmian wpływa na każdy element danych, które możemy obserwować. Pojedyncze piksele obrazu czerwonego samochodu mogą w nocy przypominać czerń. Sylwetka samochodu zależy od kąta, pod którym na nią patrzymy. Większość zastosowań wymaga od nas rozwikłania czynników zmian i odrzucenia tych, na których nam nie zależy. Oczywiście wybranie takich cech na wysokim poziomie abstrakcji z nieprzetworzonych danych może być trudne. Wiele czynników zmian, jak akcent mówcy, da się zidentyfikować tylko za pomocą wyrafinowanego rozumienia danych, na poziomie ludzkiego umysłu. Gdy uzyskanie reprezentatywnych informacji jest tak samo trudne, jak rozwiązanie problemu, wydaje się, że taka reprezentatywność na niewiele się zdaje.

Deep learning rozwiązuje ten podstawowy problem w reprezentatywnym uczeniu się dzięki wprowadzeniu reprezentatywności wyrażonej w formie innej, prostszej reprezentacji. Umożliwia komputerowi zbudowanie złożonych pojęć na podstawie prostszych. Na rysunku 1.2 pokazano, jak system deep learning może reprezentować pojęcie opisujące obraz osoby przez połączenie prostszych pojęć, jak narożniki i kontury, które z kolei są zdefiniowane jako krawędzie.

Typowym przykładem modelu deep learningu jest głęboka sieć jednokierunkowa lub wielowarstwowy perceptron (MLP). Ten ostatni to funkcja matematyczna odwzorowująca pewien zestaw danych wejściowych na war-



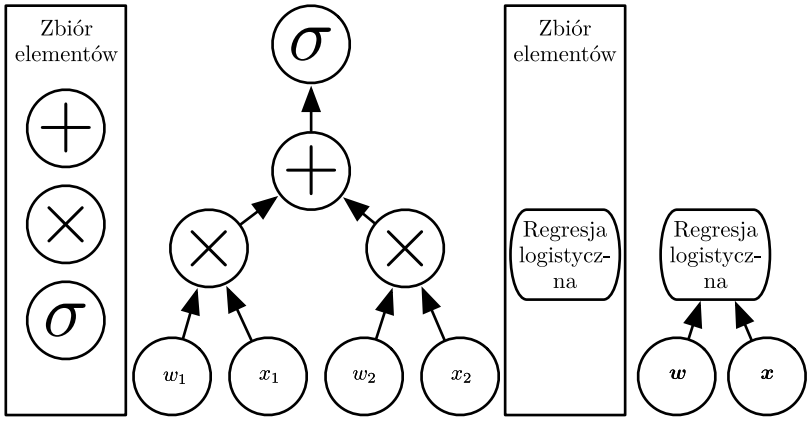
Rysunek 1.2. Ilustracja modelu deep learningu. Komputerowi trudno zrozumie¢ znaczenie surowych danych pochodz¡cych ze zmys ªów, jak obraz reprezentowany przez zbiór pikseli. Funkcja odwzorowuj¡ca zbiór pikseli na to»samo±¢ obiektu jest bardzo skomplikowana. Nauczenie si¦ lub wyznaczenie tego odwzorowania wydaje si¦ nie do pokonania w bezpo±rednim podej±ciu. Deep learning rozwi¡zuje t¦ trudno±¢ dzi¦ki podzia ªowi skomplikowanego odwzorowania na seri¦ zagnie»d»onych prostszych odwzorowa«, z których ka»de jest opisane w innej warstwie modelu. Dane wej±ciowe s¡ prezentowane w widocznej warstwie, któr¡ nazywamy tak, gdy» zawiera zmienne, które mo»na obserwowa¢. Potem nast¦puje ci¡g warstw ukrytych, wyci¡gaj¡cych z obrazu coraz bardziej abstrakcyjne cechy. Nazywa si¦ je ukrytymi, gdy» ich warto±ci nie s¡ podane w danych; model musi sam okre±li¢, które poj¦cia s¡ u»yteczne do wyt ªumaczenia zwi¡zków w obserwowanych danych. Obrazy stanowi¡ tu wizualizacj¦ pewnej cechy reprezentowanej przez ka»d¡ ukryt¡ jednostk¦. Znaj¡c piksele, pierwsza warstwa mo»e ªatwo zidenty kowa¢ kraw¦dzie, porównuj¡c jasno±¢ s¡siednich pikseli. Maj¡c opis brzegów zawarty w pierwszej warstwie, druga warstwa mo»e ªatwo odnale¹¢ naro»niki i rozszerzone kontury. Maj¡c opis obrazu z drugiej ukrytej warstwy w postaci naro»ników i konturów, trzecia ukryta warstwa mo»e wykry¢ ca ªe fragmenty okre±lonych obiektów, dzi¦ki znalezieniu specjalnego zbioru konturów i naro»ników. Wreszcie ten opis obrazu w kategorii cz¦±ci obiektu, które si¦ na niego sk ªadaj¡, mo»e by¢ wykorzystany do rozpoznania obiektów znajduj¡cych si¦ na obrazie (obrazy zamieszczono za Zeiler and Fergus 2014, za zgod¡ autorów)

tości wynikowe. Funkcja jest zbudowana z wielu mniejszych funkcji. Każde zastosowanie innej funkcji matematycznej możemy rozumieć jako nową reprezentację danych wejściowych. Cel poznania właściwej reprezentacji danych zapewnia jedną perspektywę deep learningu. Inną perspektywą jest fakt, że głębokość warstw pozwala komputerowi nauczyć się wielokrokowego programu komputerowego. Każda warstwa reprezentacji może być rozumiana jako stan pamięci komputera po równoległym wykonaniu kolejnego zbioru instrukcji. Sieci o większej głębokości mogą wykonywać więcej instrukcji sekwencyjnych. Mają one wielką moc, gdyż kolejne instrukcje mogą odwoływać się do wyników wcześniejszych instrukcji.

Zgodnie z tym spojrzeniem na deep learning nie wszystkie warstwy informacji muszą koniecznie kodować czynniki zmian, które objaśniają dane wejściowe. Reprezentacja przechowuje także informacje o stanie, co pomaga stworzyć program, który potrafi zrozumieć dane. Informacja o stanie może stanowić analogię dla licznika lub wskaźnika w tradycyjnym programie komputerowym. Nie ma nic wspólnego z zawartością konkretnych danych, ale pomaga modelowi w organizacji przetwarzania.

Są dwa sposoby pomiaru głębokości modelu. Pierwszy opiera się na liczbie kolejnych instrukcji, które należy wykonać, aby wyznaczyć architekturę problemu. Możemy to traktować jak najdłuższą ścieżkę w schemacie blokowym, który opisuje sposób obliczania każdego z wyników modelu na podstawie danych wejściowych. Podobnie dwa programy komputerowe mają różne długości, zależnie od tego, w jakim języku programowania napisano program. Funkcje można narysować w schemacie blokowym z różnym zagłębieniem, które zależy od tego, które funkcje mogą być używane w kolejnych krokach schematu. Na rysunku 1.3 widać, jak wybór języka może dać dwa różne pomiary tej samej architektury.

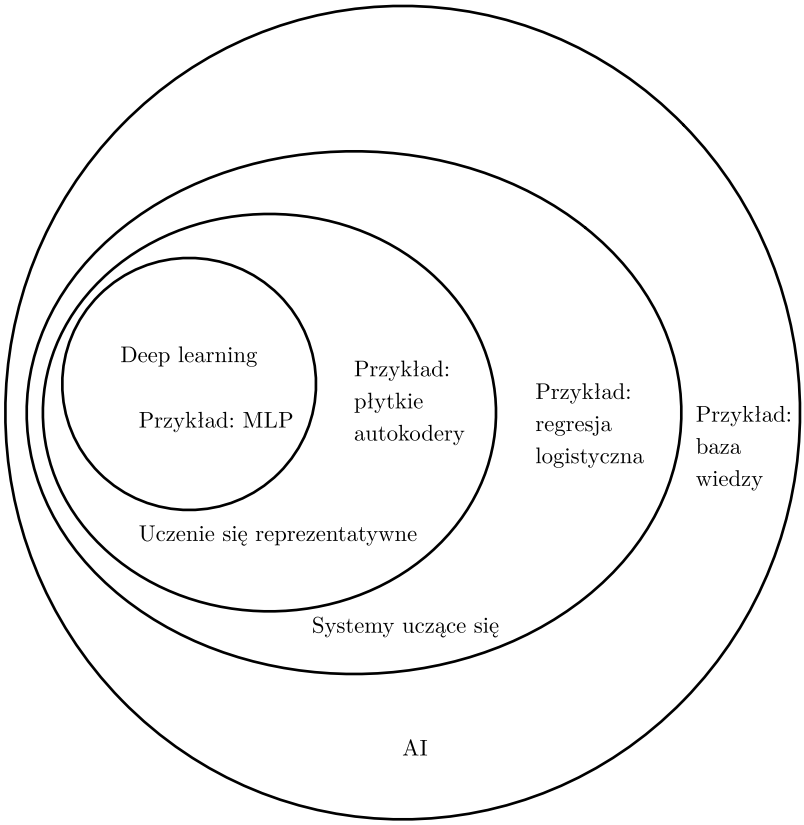
Inne podejście używane w modelach probabilistycznych dotyczy głębokości modelu nie jako głębokości grafu obliczeniowego, lecz jako głębokości grafu opisującego, jak pojęcia są ze sobą powiązane. W tym przypadku głębokość schematu blokowego obliczeń potrzebnych do wyznaczenia reprezentacji każdego z pojęć może być głębsza niż graf samych tych pojęć. Jest tak dlatego, że rozumienie przez system prostszych pojęć może być zdefiniowane przez informację o pojęciach złożonych. Na przykład system AI obserwujący obraz twarzy, gdzie jedno oko jest w cieniu, może z początku zobaczyć tylko jedno oko. Po wykryciu, że na obrazie jest twarz, system może wywnioskować, iż jest tam także drugie oko. W takim przypadku graf pojęć obejmuje tylko dwie warstwy – warstwę dla oczu i warstwę dla twarzy – ale graf obliczeniowy to $2n$ warstw, gdyż estymacja każdego pojęcia wykonywana jest n razy.



Rysunek 1.3. Grafy obliczeniowe odwzorowuj¡ce wej±cie na wyj±cie, gdzie ka»dy w¦ze³ oznacza wykonanie dzia³ania. G³¦boko±¢ to d³ugo±¢ najd³u»szej ±cie»ki od wej±cia do wyj±cia, zale»na od de nicji elementów sk³adaj¡cych si¦ na mo»liwe kroki obliczeniowe. Obliczenia pokazane na tym gra e to wyniki modelu regresji logistycznej $\sigma(\boldsymbol{w}^T\boldsymbol{x})$, gdzie σ to logistyczna funkcja sigmoidalna. Je±li dodawanie, mno»enie i logistyczne sigmoidy b¦d¡ elementami j¦zyka komputerowego, to model ten ma g³¦boko±¢ trzy. Je±li elementem jest sama regresja logistyczna, to model ma g³¦boko±¢ jeden

Ponieważ nie jest zawsze jasne, które z tych dwóch podejść – głębokość grafu komputerowego czy głębokość grafu modelu probabilistycznego – jest odpowiednia, a także dlatego, że różni ludzie wybiorą inny zbiór najmniejszych elementów, z których zbudują graf, nie ma jednej poprawnej wartości opisującej długość programu komputerowego. Nie ma też zgody, jak dużej głębokości wymaga model, aby można go było nazwać głębokim. Jednak deep learning można bezpiecznie uznać za studium modeli obejmujących większą liczbę składników poznawanych funkcji czy pojęć niż w tradycyjnych systemach uczących się.

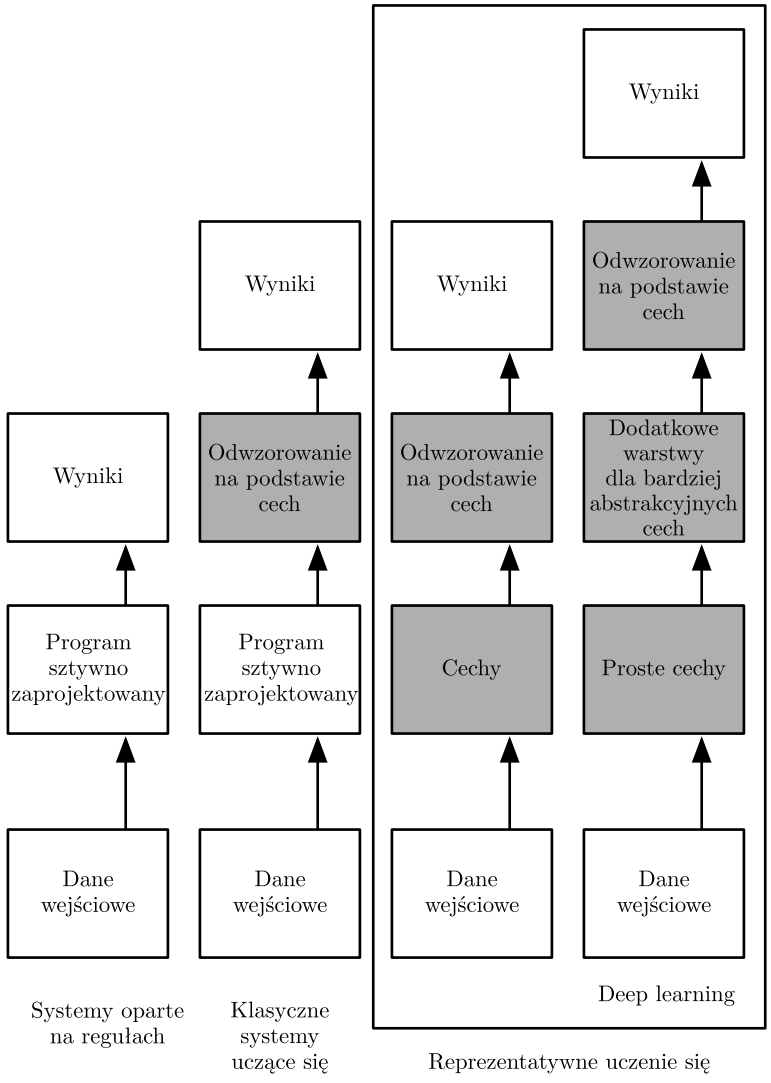
Podsumowując, deep learning, które jest tematem tej książki, stanowi podejście do AI. Konkretnie jest to rodzaj systemów uczących się, technika umożliwiająca systemom komputerowym ulepszenia oparte na doświadczeniu i danych. Systemy uczące się to jedyne sensowne podejście do budowy systemów AI, które mogą działać w skomplikowanych środowiskach świata rzeczywistego. Deep learning jest tym rodzajem systemów uczących się, który osiąga wielką moc i elastyczność, reprezentując świat jako zagnieżdżaną hierarchię pojęć, przy czym każde pojęcie jest zdefiniowane w powiązaniu z pojęciami prostszymi i bardziej abstrakcyjnymi reprezentacjami wyznaczanymi za pomocą mniej abstrakcyjnych. Rysunek 1.4 ilustruje związki między różnymi dyscyplinami AI. Rysunek 1.5 obrazuje schemat wysokiego poziomu pokazujący, jak to działa.



Rysunek 1.4. Diagram Venna pokazuj¡cy, dlaczego deep learning jest rodzajem uczenia si¦ reprezentacji, które z kolei jest typem systemu ucz¡cego si¦ u»ywanego w wielu, lecz nie wszystkich podej±ciach do AI. Diagram Venna stanowi przyk²ad technologii AI

1.1. Kto powinien przeczytać tę książkę?

Książka ta przyda się wielu czytelnikom, ale została napisana dla dwóch rodzajów odbiorców. Jedną grupą są studenci (na studiach magisterskich i doktoranckich) zdobywający wiedzę o systemach uczących się, w tym także ci, którzy chcą specjalizować się w badaniach w dziedzinie deep learningu i sztucznej inteligencji. Drugą grupą docelową są inżynierowie oprogramowania, którzy nie mają za sobą nauki z dziedziny systemów uczących się i statystyki, ale chcą szybko nadrobić zaległości i zająć się wykorzystaniem deep learningu w swojej dziedzinie. Metody te sprawdziły się w wielu dziedzinach informatyki, jak wizualizacje komputerowe, przetwarzanie głosu i dźwięku, przetwarzanie języków naturalnych, robotyka, bioinformatyka i chemia, gry wideo, mechanizmy wyszukiwania czy ogłoszenia i finanse w trybie online.



Rysunek 1.5. Schematy blokowe pokazuj¡ce ró»ne elementy systemu AI i ich wzajemne zwi¡zki z ró»nymi dziedzinami AI. Zacieniowane ramki pokazuj¡ komponenty, których mo»na nauczy¢ si¦ na podstawie danych

Książka składa się z trzech części, aby najlepiej dopasować ją do potrzeb rozmaitych czytelników. Część I obejmuje opis narzędzi matematycznych i pojęć z dziedziny systemów uczących się. Część II opisuje najważniejsze algorytmy deep learningu, które są poznanymi technikami. Część III zawiera nowe pomysły, które są powszechnie uważane za przyszłość deep learningu.

Czytelnicy mogą swobodnie opuszczać fragmenty, które są poza zakresem ich zainteresowań lub wiadomości. Osoby znające algebrę liniową, prawdopodobieństwo i podstawowe pojęcia systemów uczących się mogą na przykład pominąć część I, a ci, którzy chcą wdrożyć działający system, powinny skończyć na części II. Aby dopomóc w wyborze czytanych rozdziałów, na rysunku 1.6 pokazano schemat blokowy ogólnej budowy książki.

Zakładamy, że wszyscy czytelnicy mają pewną wiedzę z dziedziny informatyki. Przyjmujemy, że znają zasady programowania, rozumieją podstawowe zagadnienia związane z wydajnością obliczeń, teorię złożoności, podstawy rachunku różniczkowego oraz pojęcia z teorii grafów.

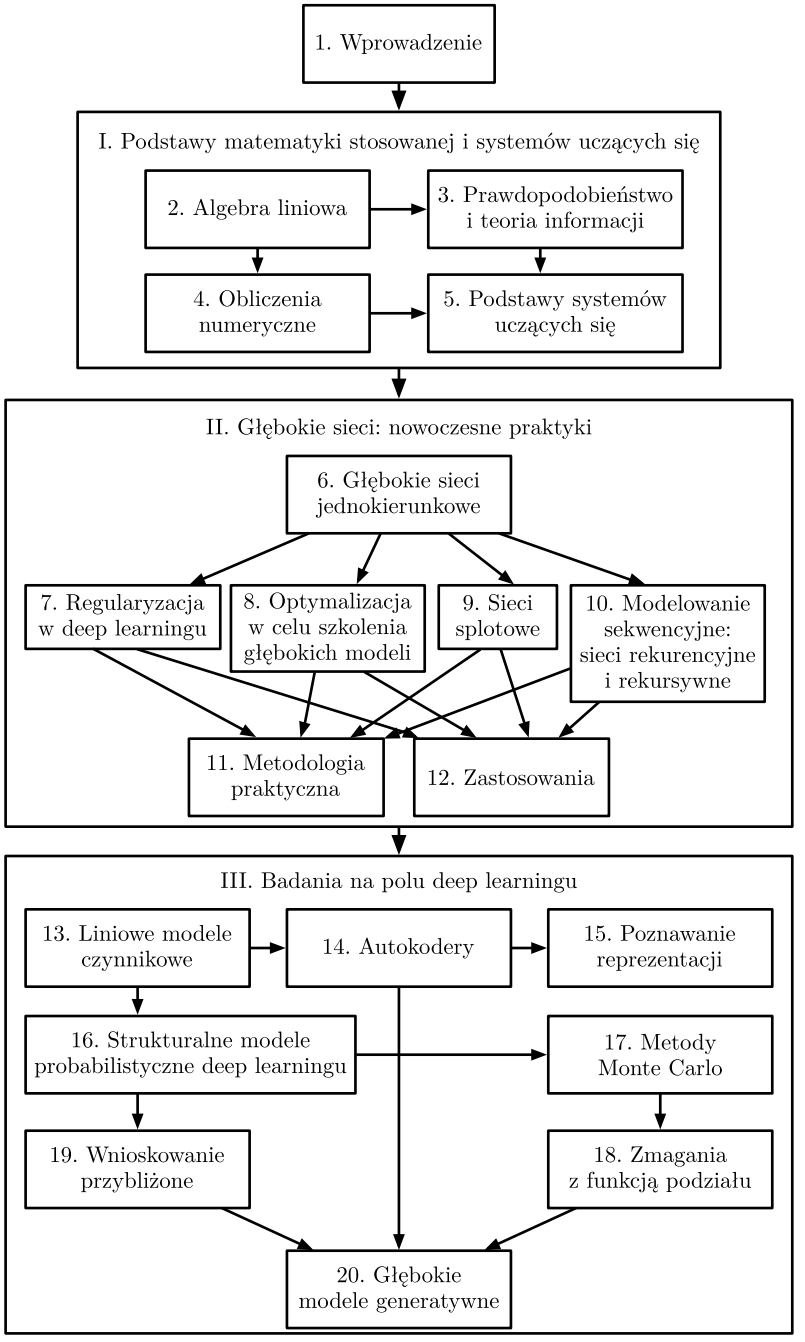
1.2. Historyczne trendy deep learningu

Najłatwiej zrozumieć deep learning w kontekście historycznym. Zamiast podawać jego szczegółową historię, poniżej wyszczególniono najważniejsze trendy:

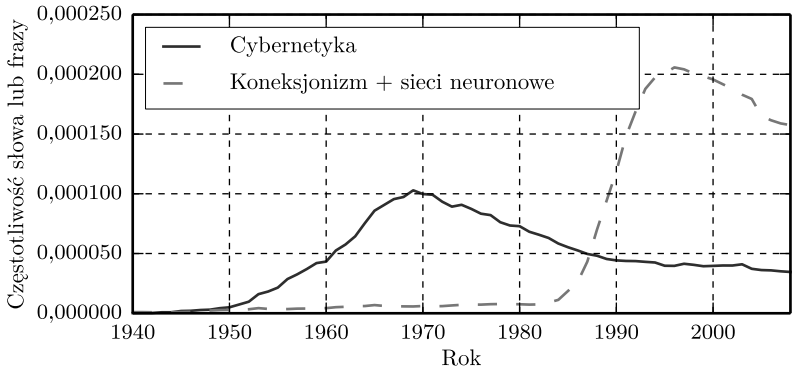
- deep learning ma długą i bogatą historię, ale znane jest pod różnymi nazwami, co odzwierciedla różne filozoficzne punkty widzenia; jego popularność pojawiała się i znikała,
- deep learning stało się bardziej użyteczne, gdy wzrosła ilość dostępnych danych szkoleniowych,
- modele deep learningu z czasem rozrosły się, a dostępna infrastruktura komputerowa (sprzęt i oprogramowanie) uległa poprawie,
- deep learning pomaga w rozwiązywaniu coraz bardziej skomplikowanych aplikacji, a dokładność jego działania wzrasta.

1.2.1. Wiele nazw i zmienne powodzenie sieci neuronowych

Wielu czytelników tej książki z pewnością słyszało o deep learningu jako o nowej pasjonującej technice, więc są zdziwieni, że jest tu mowa o historii. W rzeczywistość deep learning ma korzenie w latach czterdziestych XX wieku. Wydaje się nowością, ale przez kilka lat poprzedzających obecną popularność był mało znany. Pojawiał się pod różnymi nazwami, a określenie „deep learning" pojawiło się niedawno. Dziedzina ta wiele razy zmieniała terminologię, co odzwierciedlało wpływy różnych naukowców i rozmaite perspektywy.



Rysunek 1.6. Ogólny sposób organizacji ksi¡»ki. Strza^{a}ka z jednego rozdzia^{a}u do drugiego wskazuje, »e poprzedni rozdzia^{a} zawiera materia^{a} poprzedzaj¡cy niezb¦dny do zrozumienia kolejnego



Rysunek 1.7. Dwie historyczne fale bada« nad sieciami neuronowymi wed^{a}ug pomiaru cz¦stotliwo±ci fraz cybernetyka i koneksjonizm lub sieci neuronowe wed^{a}ug Google Botoks (trzecia fala jest zbyt nowa, »eby si¦ tu znale^{1}¢). Pierwsza fala zacz¦^{a}a si¦ od cybernetyki w latach 1940 1960, wraz z rozwojem teorii biologicznego uczenia si¦ (McCulloch and Pitts 1943, Hebb 1949) oraz implementacji pierwszych modeli, jak perceptron (Rosenblatt 1958), co pozwoli^{a}o na szkolenie pojedynczego neuronu. Druga fala zacz¦^{a}a si¦ od podej±cia koneksjonistycznego z okresu 1980 1995, z propagacj¡ wsteczn¡ (Rumelhart et al. 1986), pozwalaj¡c¡ szkoli¢ sieci neuronowe z jedn¡ lub dwiema ukrytymi warstwami. Obecna trzecia fala zacz¦^{a}a si¦ oko^{a}o roku 2006 (Hinton et al. 2006, Bengio et al. 2007, Ranzato et al. 2007), a jej opis w ksi¡»ce pojawi^{a} si¦ w roku 2016. Poprzednie dwie fale uj¦to w ksi¡»kach znacznie pó^{1}niej ni» odpowiadaj¡ce im badania naukowe

Wyczerpująca historia deep learningu wykracza poza zakres tej książki, jednak dodatkowy kontekst pomaga w zrozumieniu tej dziedziny. Zaznaczyły się trzy fale jej rozwoju: deep learning pod nazwą cybernetyki w latach 1940–1960, deep learning znane jako koneksjonizm w latach 1980–1990 oraz obecne odrodzenie już jako deep learning począwszy od roku 2006. Pokazano to na rysunku 1.7.

Jedne z pierwszych algorytmów uczenia się, które znamy do dziś, miały na celu modelowanie komputerowe biologicznych procesów uczenia się, czyli modelowały, jak to się dzieje, że uczymy się i co może się wtedy dziać w mózgu. Wynikiem była pierwsza nazwa deep learningu – sztuczne sieci neuronowe (ang. *artificial neural networks*, ANN). Odpowiadająca temu perspektywa modeli deep learningu mówi, że są to systemy inżynierskie inspirowane przez mózg biologiczny (ludzki lub zwierzęcy). Choć niektóre rodzaje sieci neuronowych używane w systemach uczących się były czasem wykorzystywane do zrozumienia działania mózgu (Hinton and Shallice 1991), nie są zwykle zaprojektowane jako realne modele funkcji biologicznych. Perspektywa neuronowa deep learningu opiera się na dwóch założeniach. Jednym z nich jest twierdzenie, że mózg stanowi dowód na to, iż możliwe jest inteligentne

zachowanie i stosunkowo prostą drogą do zbudowania inteligencji jest opracowanie komputerowych zasad działania mózgu i powielanie jego działania. Inną perspektywą jest przyjęcie, że ciekawe byłoby zrozumienie mózgu i zasad rządzących ludzką inteligencją, aby modele systemów uczących się rzucające światło na te naukowe pytania były użyteczne niezależnie od ich zdolności do rozwiązania zastosowań inżynierskich.

Nowoczesny termin „deep learning" wychodzi poza neuronaukową perspektywę dotyczącą obecnej generacji modeli dla systemów uczących się. Odwołuje się do bardziej ogólnej zasady uczenia się na różnych poziomach złożoności, co można zastosować w schematach systemów uczących się, które nie muszą być inspirowane przez neurony.

Najwcześniejsze modele poprzedzające nowoczesne deep learningu stanowiły modele liniowe motywowane perspektywą neuronaukową. Modele te były zaprojektowane jako pobierające dane wejściowe w postaci zbioru n wartości $x_1, \ldots, x_n$ i wiążące je z wynikami w postaci wyników y. Modele te uczą się, jaki jest zbiór wag $w_1, \ldots, w_n$, i obliczają wyniki według funkcji postaci $f(\boldsymbol{x}, \boldsymbol{w}) = x_1 w_1 + \cdots + x_n w_n$. Ta pierwsza fala sieci neuronowych była znana jako cybernetyka, jak to pokazano na rysunku 1.7.

Model neuronów McCullocha-Pittsa (McCulloch and Pitts 1943) był pierwszym, liniowym modelem funkcji mózgu. Potrafił on rozróżnić dwa rodzaje danych wejściowych, sprawdzając czy wartość funkcji $f(\boldsymbol{x}, \boldsymbol{w})$ jest dodatnia, czy ujemna. Oczywiście, aby model odpowiadał pożądanym definicjom kategorii danych, wagi musiały być poprawnie ustawione. Mógł to robić człowiek. W lata pięćdziesiątych XX wieku pierwszym modelem, który potrafił uczyć się wag definiujących kategorie na podstawie przykładów danych w każdej z kategorii, był perceptron (Rosenblatt 1958, 1962). **Adaptacyjny liniowy element** (ADALINE) pochodzący z tego samego okresu po prostu zwracał wartość funkcji $f(\boldsymbol{x})$, aby przewidzieć liczbę rzeczywistą (Widrow and Hoff 1960), i potrafił nauczyć się przewidywać te wartości na podstawie danych.

Te proste algorytmy uczące się wywarły duży wpływ na dzisiejszy krajobraz systemów uczących się. Algorytm szkoleniowy używany do adaptacji wag w ADALINE stanowił szczególny przypadek algorytmu określanego jako algorytm **metodą gradientu stochastycznego**. Nieco zmodyfikowana wersja tego algorytmu dominuje w dzisiejszych modelach szkoleniowych w deep learningu.

Modele oparte na $f(\boldsymbol{x}, \boldsymbol{w})$, używane przez perceptron i ADALINE, są określane jako **modele liniowe**. Nadal są najszerzej używanymi modelami w systemach uczących się, choć w wielu przypadkach szkolenie przebiega w inny sposób, niż miało to miejsce w oryginalnych wersjach modelu.

Modele liniowe mają wiele ograniczeń. Najbardziej znanym jest fakt, że nie mogą nauczyć się funkcji XOR, gdzie $f([0,1], \boldsymbol{w}) = 1$ i $f([1,0], \boldsymbol{w}) = 1$, ale $f([1,1], \boldsymbol{w}) = 0$ i $f([0,0], \boldsymbol{w}) = 0$. Krytycy, którzy widzą te wady modeli liniowych, spowodowali ogólne odejście od uczenia się inspirowanego biologicznie (Minsky and Papert 1969). Było to pierwsze tąpnięcie popularności sieci neuronowych. Dziś neuronauka jest uznawana za ważne źródło inspiracji dla badaczy deep learningu, ale nie stanowi podstawowej wskazówki w tej dziedzinie.

Podstawową przyczyną mniejszej roli neuronauki w obecnych badaniach nad deep learningiem jest po prostu fakt, że nie mamy dość informacji o mózgu, które mogą stanowić wskazówkę. Aby dojść do głębokiego zrozumienia prawdziwych algorytmów używanych w mózgu, powinniśmy mieć możliwość jednoczesnego monitorowania aktywności tysięcy (przynajmniej jako minimum) połączonych ze sobą neuronów. Ponieważ nie możemy tego zrobić, jesteśmy nadal daleko od zrozumienia nawet najprostszych i najlepiej poznanych części mózgu (Olshausen and Field 2005).

Neuronauka dała nadzieję na to, że jeden algorytm deep learningu umie rozwiązać wiele różnych zadań. Neuronaukowcy odkryli, że fretki potrafią nauczyć się „widzieć" za pomocą obszaru mózgu przetwarzającego dźwięki, jeśli ich mózgi zostaną zmienione tak, aby sygnały wzrokowe były przesyłane do tego obszaru (Von Melchner et al. 2000). Sugeruje to, że duża część mózgu ssaków może korzystać z jednego algorytmu do rozwiązywania w mózgu różnych zadań. Zanim powstała ta hipoteza, badania nad systemami uczącymi się były podzielone, a różne społeczności naukowców zajmowały się przetwarzaniem języka naturalnego, wizją, planowaniem ruchu i rozpoznawaniem mowy. Dziś te społeczności nadal pracują oddzielnie, ale wszystkie grupy badające deep learningu badają jednocześnie wiele lub wszystkie te dziedziny zastosowań.

Możemy nakreślić ogólne wskazówki na bazie neuronauki. Podstawowa zasada wielu jednostek obliczeniowych, które stają się inteligentne tylko dzięki wzajemnym połączeniom, jest inspirowana działaniem mózgu. *Neocognitron* (Fukushima 1980) wprowadza potężną architekturę modelu przetwarzania obrazów zainspirowaną strukturą systemu wzrokowego ssaków i stał się podstawą nowoczesnych sieci splotowych (LeCun et al. 1998b), jak to pokazano w punkcie 9.10. Większość dzisiejszych sieci neuronowych opiera się na modelu znanym jako **poprawiona jednostka liniowa** (ang. *rectified linear unit*, ReLU). Oryginalny *cognitron* (Fukushima 1975) wprowadził bardziej skomplikowaną wersję w dużym stopniu inspirowaną naszą wiedzą o działaniu mózgu. Uproszczona współczesna wersja została opracowana dzięki połączeniu pomysłów z różnych punktów widzenia, przy czym Nair i Hinton

(2010) oraz Glorot et al. (2011a) podają neuronaukę jako źródło wpływu, a Jarrett et al. (2009) powołują się na wpływy bardziej inspirowane wiedzą inżynierską. Wprawdzie neuronauka jest ważnym źródłem inspiracji, jednak nie powinna być traktowana jako ścisłe wskazówki. Wiemy, że pojedyncze neurony obliczają całkiem inne funkcje niż nowoczesne jednostki ReLU, lecz większy realizm w tym zakresie wcale nie doprowadza do lepszej wydajności systemów uczących się. Poza tym, choć neuronauka z powodzeniem zainspirowała szereg architektur sieci neuronowych, nadal nie mamy dość wiedzy o biologicznym uczeniu się, aby neuronauka dawała wiele wskazówek dla algorytmów uczących się używanych przez nas do szkolenia.

Media często podkreślają podobieństwo deep learningu do pracy mózgu. Wprawdzie prawdą jest, że pracujący nad deep learningiem podkreślają wpływ sposobu działania mózgu bardziej niż badacze w innych dziedzinach systemów uczących się, jak w przypadku jądra maszyn lub statystyk bayesowskich, ale nie należy widzieć deep learningu jako próby symulacji mózgu. Nowoczesny deep learning czerpie inspirację z wielu dziedzin, jak matematyka stosowana, w tym algebra liniowa, prawdopodobieństwo, a także teoria informacji i optymalizacja numeryczna. Niektórzy badacze deep learningu przytaczają neuronaukę jako istotne źródło inspiracji, inni wcale nie poruszają tego tematu.

Warto zauważyć, że wysiłek wkładany w zrozumienie, jak mózg pracuje na poziomie algorytmicznym, jest nadal żywy. Te starania są znane jako „komputerowa neuronauka" i stanowią oddzielne pole badań w stosunku do deep learningu. Badacze często przechodzą między tymi dwiema dziedzinami. Deep learning koncentruje się przede wszystkim na tym, jak zbudować systemy komputerowe, które potrafią z powodzeniem rozwiązywać zadania wymagające inteligencji, a komputerowa neuronauka zajmuje się budowaniem dokładniejszych modeli pracy mózgu.

W latach osiemdziesiątych XX wieku pojawiła się druga fala badań nad sieciami neuronowymi, głównie dzięki ruchowi zwanemu **koneksjonizmem** lub **równoległym przetwarzaniem rozproszonym** (Rumelhart et al. 1986c, McClelland et al. 1995). Koneksjonizm powstał w kontekście wiedzy kognitywnej. Jest to interdyscyplinarne podejście do zrozumienia umysłu, łączące wiele różnych poziomów analizy. We wczesnych latach osiemdziesiątych większość kognitywistów analizowała modele symbolicznego rozumowania. Mimo ich popularności modele symboliczne były trudne do wyjaśnienia w sensie zrozumienia, jak mózg może je wdrażać, używając neuronów. Koneksjoniści zaczęli analizować modele poznawcze, które mogły mieć źródło w implementacjach neuronowych (Touretzky and Minton 1985), ożywiając wiele pomysłów pochodzących z prac psychologa Donalda Hebba z lat czterdziestych (Hebb 1949).

Podstawowa idea koneksjonizmu stanowi, że duża liczba prostych elementów obliczeniowych może razem w sieci osiągnąć inteligentne zachowanie. To spostrzeżenie odnosi się zarówno do neuronów w systemach biologicznych, jak i do ukrytych jednostek w modelach obliczeniowych.

W czasach rozwoju ruchu koneksjonalistów z lat osiemdziesiątych pojawiło się kilka kluczowych pojęć, które do dziś pozostają w centrum deep learningu. Jednym z nich jest **rozproszona reprezentacja** (Hinton et al. 1986). Mówi ona, że każde dane wejściowe do systemu powinny być reprezentowane przez wiele cech, a każda z nich powinna być częścią reprezentacji wielu możliwych danych wejściowych. Przypuśćmy na przykład, że mamy system wizyjny, który potrafi rozróżniać samochody, ciężarówki i ptaki, a obiekty te mogą być czerwone, zielone lub niebieskie. Jednym ze sposobów prezentacji danych wejściowych może być oddzielny neuron lub ukryta jednostka aktywizująca każdą z dziewięciu możliwych kombinacji: czerwona ciężarówka, czerwony ptak, czerwony samochód, zielona ciężarówka i tak dalej. Wymaga to dziewięciu różnych neuronów, a każdy z nich musi niezależnie nauczyć się pojęcia koloru i tożsamości obiektu. Sposobem poprawy tej sytuacji jest skorzystanie z rozproszonej reprezentacji, gdzie trzy neurony opisują kolor, a trzy tożsamość obiektu. Wymaga to tylko sześciu neuronów zamiast dziewięciu, a neuron opisujący czerwień może dowiedzieć się o tym kolorze na podstawie obrazów samochodów, ciężarówek i ptaków, nie zaś z obrazów jednego typu obiektów. Pojęcie rozproszonej reprezentacji stanowi istotę tej książki i jest w szczegółach opisane w rozdziale 15.

Innym istotnym osiągnięciem ruchu koneksjonistycznego jest udane zastosowanie wstecznej propagacji w celu szkolenia głębokich sieci neuronowych za pomocą wewnętrznych reprezentacji i popularyzacja algorytmu propagacji wstecznej (Rumelhart et al. 1986a, LeCun 1987). Ten algorytm zyskiwał i tracił na popularności, ale w czasie pisania tej książki stanowi dominujące podejście do modeli deep learningu.

W latach dziewięćdziesiątych XX wieku badacze poczynili znaczące postępy w modelowaniu sekwencji z wykorzystaniem sieci neuronowych. Hochreiter (1991) i Bengio et al. (1994) rozwiązali niektóre podstawowe trudności matematyczne przy modelowaniu długich sekwencji, opisane w punkcie 10.7. Hochreiter i Schmidhuber (1997) wprowadzili sieć z długą pamięcią krótkoterminową (ang. *long short-term memory*, LSTM) do rozwiązywania niektórych z tych trudności. Dziś LSTM jest szeroko wykorzystywana w wielu sekwencyjnych zadaniach modelowania, w tym przy przetwarzaniu języka naturalnego w Google.

Druga fala badań nad sieciami neuronowymi trwała do połowy lat dziewięćdziesiątych. Działania opierające się na sieciach neuronowych i innych

technologiach AI zaczęły brnąć w nierealistyczne obietnice, szukając inwestorów. Gdy nie spełniły tych nierealnych oczekiwań, inwestorzy byli zawiedzeni. Jednocześnie w innych dziedzinach systemów uczących się dokonywano postępów. Maszyny oparte na jądrze (ang. *kernel machines*; Boser et al. 1992, Cortes and Vapnik 1995, Schölkopf et al. 1999) oraz modele graficzne (Jordan 1998) osiągnęły dobre wyniki w wielu ważnych zadaniach. Te dwa czynniki doprowadziły do spadku popularności sieci neuronowych, co trwało do roku 2007.

W tym czasie sieci neuronowe nadal pozwalały uzyskać świetną wydajność przy wykonywaniu niektórych zadań (LeCun et al. 1998b, Bengio et al. 2001). Kanadyjski instytut CIFAR (Canadian Institute for Advanced Research) pomógł w utrzymaniu badań nad sieciami neuronowymi poprzez swoją inicjatywę badawczą NCAP (Neural Computation and Adaptive Perception). Program ten zjednoczył zespoły badawcze pracujące nad systemami uczącymi się pod kierunkiem Geoffreya Hintona z Uniwersytetu w Toronto, Yoshuy Bengio z Uniwersytetu w Montrealu oraz Yanna LeCun z Uniwersytetu w Nowym Jorku. Multidyscyplinarne badania z inicjatywy CIFAR NCAP obejmowały neuronaukowców oraz ekspertów z zakresu widzenia ludzkiego i komputerowego.

Obecne głębokie sieci miały wówczas opinię trudnych do nauczenia. Teraz wiemy, że algorytmy dostępne od lat dziewięćdziesiątych działają dość dobrze, ale około roku 2006 nie było to oczywiste. Problem polega zapewne na tym, że algorytmy te są zbyt kosztowne obliczeniowo, aby można było pozwolić sobie na eksperymenty na dostępnym wówczas sprzęcie.

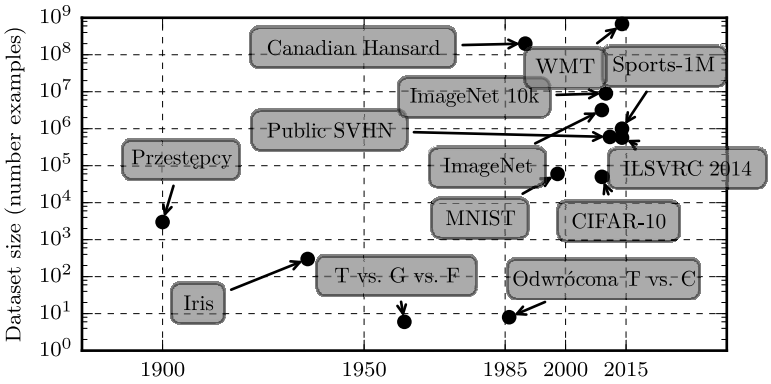
Trzecia fala badań nad sieciami neuronowymi zaczęła się od przełomu z roku 2006. Geoffrey Hinton pokazał rodzaj sieci neuronowej o nazwie *głęboka sieć przekonań*, która mogła być szkolona za pomocą strategii zachłannego wstępnego szkolenia opartego na warstwach (Hinton et al. 2006), co szczegółowo jest omawiane w punkcie 15.1. Inna grupa badawcza powiązana z CIFAR szybko pokazała, że ta sama strategia może być wykorzystana do wielu innych rodzajów sieci (Bengio et al. 2007, Ranzato et al. 2007a) i systematycznie pomagała w poprawie uogólnienia jej na przykładach testowych. Ta fala badań nas sieciami neuronowymi spopularyzowała określenie „deep learning", aby podkreślić, że badacze potrafią teraz głębiej szkolić sieci neuronowe, niż to było możliwe wcześniej, oraz aby skoncentrować uwagę na tym, jak teoretycznie ważna jest głębokość (Bengio and LeCun 2007, Delalleau i Bengio 2011, Pascanu et al. 2014a, Montufar et al. 2014). W tamtym czasie głębokie sieci neuronowe przewyższyły inne systemy AI, które opierają się na innych technikach uczenia się, a także opracowane ręcznie funkcje. Ta trzecia fala popularności sieci neuronowych trwa do dziś, choć główny cel badań nad deep learningiem znacznie zmieniły się z upływem czasu. Trzecia

fala rozpoczęła się od koncentracji na nowych nienadzorowanych technikach uczenia się i na zdolności głębokich modeli do generalizowania na podstawie małych zbiorów danych, lecz dziś nie ma już zainteresowania starszymi nadzorowanymi algorytmami uczenia się i zdolnością głębokich modeli do stosowania dużych etykietowanych zbiorów danych.

1.2.2. Rosnące rozmiary zbiorów danych

Można się zastanawiać, dlaczego deep learning dopiero ostatnio uznano za ważną technikę, choć pierwsze eksperymenty ze sztucznymi sieciami neuronowymi prowadzono już w latach pięćdziesiątych XX wieku. Deep learning był z powodzeniem używany w zastosowaniach komercyjnych od lat dziewięćdziesiątych, ale często uważany za sztukę, a nie technikę; twierdzono też, że do obsługi wymaga eksperta. To prawda, że dobra wydajność algorytmu wymaga posiadania umiejętności. Na szczęście ich zakres maleje wraz ze wzrostem ilości danych szkoleniowych. Algorytmy uczenia się, które z trudem rozwiązywały banalnie proste problemy w latach osiemdziesiątych, dzięki modelom szkoleniowym przeszły znaczące zmiany upraszczające szkolenia w przypadku bardzo głębokich architektur. Najważniejszą nowością jest fakt, że dziś istnieją algorytmy o zasobach wystarczających do odniesienia sukcesu. Na rysunku 1.8 pokazano, jak wielkość wzorcowych zbiorów danych rozrosła się z upływem czasu. Ten trend jest wynikiem rosnącej digitalizacji społeczeństwa. W miarę jak coraz więcej naszych działań odbywa się na komputerach, coraz więcej naszych działań jest rejestrowanych.

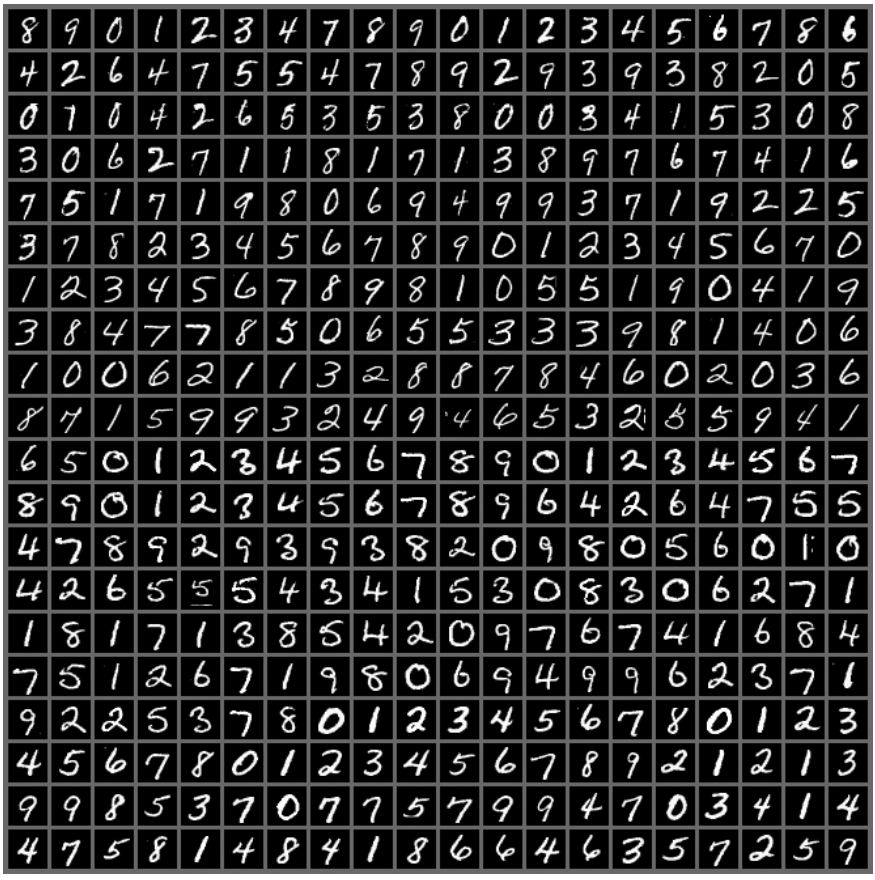
Nasze komputery są w coraz większym stopniu połączone z siecią, więc łatwiej scentralizować zapisy i dołączyć je do zbioru danych odpowiedniego dla aplikacji systemów uczących się. Czasy Big Data sprawiły, że systemy uczące się są prostsze, gdyż podstawowy ciężar estymacji statystycznej – uogólnianie nowych danych na podstawie obserwacji ich niewielkich ilości – stał się znacząco mniejszy. W roku 2016 zasadą jest, że nadzorowany algorytm deep learning osiąga akceptowalną wydajność przy około 5000 etykietowanych przykładach w danej kategorii i potrafi doścignąć lub przewyższyć wydajność ludzi, jeśli jest szkolony na podstawie nie mniej niż 10 milionów etykietowanych przykładów. Powodzenie w działaniu przy zbiorach danych mniejszych niż te wielkości stanowi ważny obszar badawczy, skupiając się w szczególności na możliwości korzystania z dużej liczby nieetykietowanych przykładów przy uczeniu się nienadzorowanym lub częściowo nadzorowanym.



Rysunek 1.8. Rosn¡ce z upªywem czasu zbiory danych. Na pocz¡tku XX wieku statystycy analizowali zbiory danych za pomoc¡ setek lub tysi¦cy r¦cznie utworzonych miar (Garson 1900, Gosset 1908, Anderson 1935, Fisher 1936). Od 1950 roku do lat osiemdziesi¡tych XX wieku pionierzy systemów ucz¡cych si¦ inspirowanych biologicznie cz¦sto pracowali na maªych syntetycznych zbiorach danych, jak mapy bitowe liter o niewielkiej rozdzielczo±ci, które byªy projektowane tak, aby zmniejszy¢ koszty obliczeniowe i pokaza¢, »e sieci neuronowe mog¡ nauczy¢ si¦ okre±lonych funkcji (Widrow and Ho 1960, Rumelhart et al. 1986b). W latach osiemdziesi¡tych i dziewi¦¢dziesi¡tych XX wieku systemy ucz¡ce si¦ zacz¦ªy obejmowa¢ wi¦cej statystyki i korzysta¢ z wi¦kszych zbiorów danych zawieraj¡cych tysi¡ce przykªadów, jak na przykªad zbiór danych MNIST (pokazany na rysunku 1.9) zawieraj¡cy skany r¦cznie pisanych liczb (LeCun et al. 1998b). W pierwszej dekadzie XXI wieku zacz¦ªy by¢ tworzone bardziej skomplikowane zbiory danych tych samych rozmiarów, jak CIFAR-10 (Krizhevsky i Hinton 2009). Pod koniec dekady i w pierwszej poªowie drugiej dekady XXI wieku znacz¡co wi¦ksze zbiory danych, zawieraj¡ce setki tysi¦cy do dziesi¡tek milionów przykªadów, caªkowicie zmieniªy mo»liwo±ci deep learningu. Te zbiory danych obejmowaªy zbiór danych Street View House Numbers (Netzer et al. 2011), ró»ne wersje zbioru danych ImageNet (Deng et al. 2009, 2010a; Russakovsky et al. 2014a) oraz zbiór danych Sports-1M (Karpathy et al. 2014). Na górze wykresu widzimy, »e zbiory danych przetªumaczonych zda«, jak zbiór IBM zbudowany na podstawie Canadian Hansard (Brown et al. 1990) oraz zbiory danych WMT 2014 English to French (Schwenk 2014), znacz¡co wyprzedzaj¡ wielko±¢ innych zbiorów danych

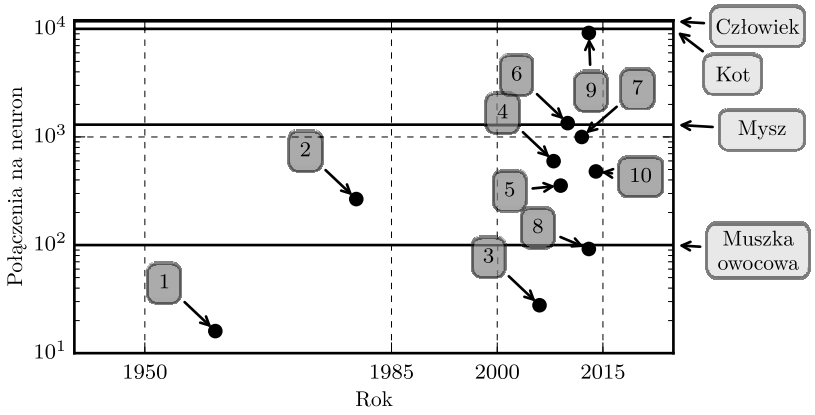
1.2.3. Rosnące rozmiary modeli

Innym ważnym powodem dzisiejszego sukcesu sieci neuronowych w porównaniu z umiarkowanym powodzeniem w latach dziewięćdziesiątych jest dostępność zasobów pozwalających na wykonywanie znacznie większych modeli. Jedną z głównych tez koneksjonizmu jest twierdzenie, że zwierzęta stają się inteligentne, gdy wiele z ich neuronów współpracuje ze sobą. Pojedynczy neuron lub niewielka ich grupa nie są szczególnie przydatne.



Rysunek 1.9. Przyk^{a}ad danych wej±ciowych ze zbioru danych MNIST. Skrót NIST oznacza National Institute of Standards and Technology (Narodowy Instytut Standardów i Technologii), instytucj¦, która zgromadzi^{a}a te dane. Litera M pochodzi od s^{a}owa mody kowany , gdy» dane zosta^{a}y wst¦pnie przetworzone, aby ^{a}atwiej by^{a}o je wykorzysta¢ w algorytmach systemów ucz¡cych si¦. Zbiór danych MNIST sk^{a}ada si¦ ze skanów r¦cznie pisanych cyfr oraz zwi¡zanych z nimi etykiet opisuj¡cych, która z cyfr od 0 do 9 znajduje si¦ na obrazku. Ten prosty sposób klasy kacji jest najprostszym i najcz¦±ciej u»ywanym testem stosowanym w badaniach nad deep learningiem. Pozostaje popularny, mimo »e jak na dzisiejsze techniki jest stosunkowo ^{a}atwy do rozwi¡zania. Geo rey Hinton opisa^{a} go jako owocówk¦ systemów ucz¡cych si¦ , co oznacza, »e pozwala badaczom systemów ucz¡cych si¦ na analiz¦ ich algorytmów w kontrolowanych warunkach laboratoryjnych, podobnie jak biolodzy badaj¡ muszki owocowe

Neurony biologiczne nie mają zbyt gęstej sieci połączeń. Jak pokazano na rysunku 1.10, nasze modele systemów uczących się mają liczbę połączeń na neuron od długiego czasu o rząd wielkości większą niż mózgi ssaków.

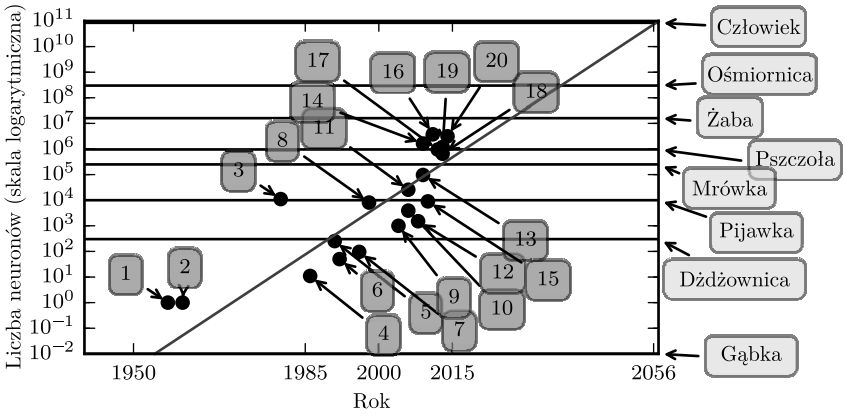


Rysunek 1.10. Liczba poª¡cze« na neuron wzgl¦dem czasu. Dawniej liczb¦ poª¡cze« mi¦dzy neuronami w sztucznych sieciach neuronowych ograniczaªy mo»liwo±ci sprz¦towe. Dzi± liczba poª¡cze« mi¦dzy neuronami wynika gªównie z zaªo»e« projektowych. Niektóre sztuczne sieci neuronowe maj¡ niemal tyle poª¡cze« na neuron, ile mniejsze ssaki, np. myszy. Nawet ludzki mózg nie ma tak wielkiej liczby poª¡cze« na neuron. Wielko±ci sieci biologicznych podano za Wikipedi¡ (2015):

1. Adaptacyjny element liniowy (*linear element*, Widrow and Hoff 1960)
2. Neokognitron (Fukushima 1980)
3. Sieć splotowa przyśpieszana przez GPU (Chellapilla et al. 2006)
4. Głęboka maszyna Boltzmanna (Salakhutdinov and Hinton 2009a)
5. Nienadzorowana sieć splotowa (Jarrett et al. 2009)
6. Wielowarstwowy perceptron przyśpieszany przez GPU (Ciresan et al. 2010)
7. Rozproszony autokoder (Le et al. 2012)
8. Sieć splotowa z wieloma GPU (Krizhevsky et al. 2012)
9. Nienadzorowana sieć splotowa COTS HPC (Coates et al. 2013)
10. GoogLeNet (Szegedy et al. 2014a)

W kontekście liczby neuronów sieci neuronowe do niedawna były niezwykle małe. Zmieniło się to dopiero ostatnio, co pokazuje rysunek 1.11, gdzie widać rozrastanie się sieci neuronowych w funkcji czasu.

Od czasu wprowadzenia ukrytych jednostek sztuczne sieci neuronowe podwajają swoje rozmiary średnio co 2,4 roku. Ten wzrost wynik z użycia szybszych komputerów z większą pamięcią oraz dostępu do większych zbiorów danych. Większe sieci mogą osiągać większą dokładność bardziej złożonych zadań. Wygląda na to, że ten trend utrwalił się na lata. Jeśli nowe techniki nie pozwolą na szybsze skalowanie, sztuczne sieci neuronowe nie będą miały tylu neuronów, jak ludzki mózg, co najmniej do lat pięćdziesiątych XXI wieku. Neurony biologiczne mogą reprezentować bardziej skomplikowane funkcje niż obecne neurony sztuczne, więc biologiczne sieci neuronowe mogą być nawet większe, niż to wynika z rysunku.



Rysunek 1.11. Wzrost rozmiaru sieci neuronowych wzgl¦dem czasu. Od czasu wprowadzenia ukrytych jednostek sztuczne sieci neuronowe podwajaj¡ swoje rozmiary mniej wi¦cej co 2,4 roku. Oto rozmiary sieci neuronowych wg Wikipedii (2015):

1. Perceptron (Rosenblatt 1958, 1962)
2. Adaptacyjny element liniowy (*linear element*, Widrow and Hoff 1960)
3. Neocognitron (Fukushima 1980)
4. Wczesne sieci propagacji wstecznej (Rumelhart et al. 1986b)
5. Rekurencyjna sieć neuronowa rozpoznawania mowy (Robinson and Fallside 1991)
6. Wielowarstwowy perceptron rozpoznawania mowy (Bengio et al. 1991)
7. Sigmoidalna sieć średniego pola przekonań (Saul et al. 1996)
8. LeNet-5 (LeCun et al. 1998b)
9. Sieć stanu echa (Jaeger and Haas 2004)
10. Sieć głębokich przekonań (Hinton et al. 2006)
11. Sieć splotowa przyśpieszana przez GPU (Chellapilla et al. 2006)
12. Głęboka maszyna Boltzmanna (Salakhutdinov and Hinton 2009a)
13. Głęboka sieć przekonań przyśpieszana przez GPU (Raina et al. 2009)
14. Nienadzorowana sieć splotowa (Jarrett et al. 2009)
15. Wielowarstwowy perceptron przyśpieszany przez GPU (Ciresan et al. 2010)
16. Sieć OMP-1 (Coates and Ng 2011)
17. Rozproszony autokoder (Le et al. 2012)
18. Sieć splotowa z wieloma GPU (Krizhevsky et al. 2012)
19. Nienadzorowana sieć splotowa COTS HPC (Coates et al. 2013)
20. GoogLeNet (Szegedy et al. 2014a).

Patrząc retrospektywnie, możemy uznać, że nie jest szczególnie zaskakujące, iż sieci neuronowe, mające mniej neuronów niż pijawki, nie potrafiły rozwiązywać skomplikowanych problemów sztucznej inteligencji. Nawet dzisiejsze sieci, dość duże z obliczeniowego punktu widzenia, mają system nerwowy mniejszy niż względnie prymitywne zwierzęta, takie jak żaba.

Powiększanie się modeli z upływem czasu, dostępność szybszych procesorów, nadejście GPU ogólnego zastosowania (opisane w punkcie 12.1.2), szybsze połączenia sieciowe i lepsza infrastruktura programowa do celów

rozproszonych obliczeń – to najważniejsze trendy w historii deep learningu. Spodziewamy się, że taki trend utrzyma się w przyszłości.

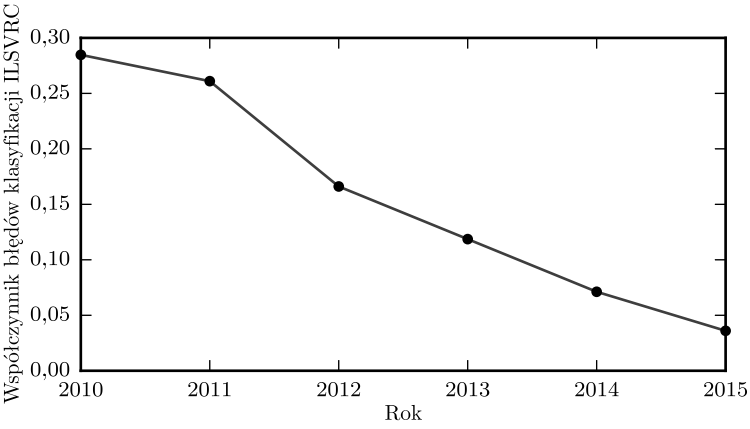
1.2.4. Rosnąca dokładność, złożoność i wpływ świata rzeczywistego

Od lat osiemdziesiątych XX wieku deep learning wciąż poprawia swoją zdolność do dokładnego rozpoznawania i przewidywania. Co więcej, jest on stale z powodzeniem stosowany do coraz szerszego spektrum zastosowań.

Najwcześniejsze modele deep learningu były używane do rozpoznawania pojedynczych obiektów na dobrze przyciętych bardzo małych obrazach (Rumelhart et al. 1986a). Od tego czasu następuje stały wzrost wielkości obrazów, które mogą być przetwarzane przez sieci neuronowe. Nowoczesne sieci rozpoznawania obiektów przetwarzają pełne fotografie w wysokiej rozdzielczości i nie wymagają, aby zdjęcie było przycięte wokół rozpoznawanego obiektu (Krizhevsky et al. 2012). Pierwsze sieci mogły rozpoznawać tylko dwa rodzaje obiektów (lub w niektórych przypadkach brak jakiegoś typu obiektu), a nowoczesne sieci zwykle rozpoznają co najmniej 1000 różnych rodzajów obiektów. Największy konkurs rozpoznawania obiektów to odbywający się co roku ImageNet Large Scale Visual Recognition Challenge (ILSVRC). Momentem błyskawicznego wzrostu deep learningu był dzień, gdy sieć splotowa po raz pierwszy wyraźnie wygrała konkurs, zmniejszając współczynnik 5 górnych błędów z 26,1% na 15,3% (Krizhevsky et al. 2012), co oznaczało, że ta sieć splotowa tworzy listę rankingową możliwych kategorii dla każdego obrazu, a poprawna kategoria pojawia się wśród pierwszych pięciu pozycji we wszystkich testach, poza 15,3%. Od tego czasu te konkursy są stale wygrywane przez głębokie sieci splotowe, a w momencie pisania tej książki postępy w rozwoju deep learningu sprowadziły procent błędnych odpowiedzi w górnych pięciu do 3,5%, jak to pokazano na rysunku 1.12.

Deep learning ma również ogromny wpływ na rozpoznawanie mowy. Po znacznej poprawie w latach dziewięćdziesiątych współczynnik błędów w rozpoznawaniu mowy zatrzymał się około roku 2000. Wprowadzenie deep learningu (Dahl et al. 2010, Deng et al. 2010b, Seide et al. 2011, Hinton et al. 2012a) do procesu rozpoznawania mowy sprawiło, że niektóre współczynniki błędów spadły o połowę. Ten temat jest bardziej szczegółowo omówiony w punkcie 12.3.

Głębokie sieci odniosły też spektakularny sukces przy wykrywaniu pieszych oraz segmentacji obrazów (Sermanet et al. 2013, Farabet et al. 2013, Couprie et al. 2013) i osiągnęły nadludzką wydajność w rozpoznawaniu znaków drogowych.



Rysunek 1.12. Zmniejszanie współczynnika błędów względem czasu. Od czasu, gdy głębokie sieci osiągnęły skalę potrzebną do konkurowania w ImageNet Large Scale Visual Recognition Challenge, co roku wygrywają ten konkurs, za każdym razem osiągając niższy współczynnik błędu. Dane za Russakovsky et al. 2014b i He et al. 2015

W czasie gdy wzrosła skala i dokładność głębokich sieci, wzrosła także złożoność zadań, które potrafią one rozwiązać. Goodfellow et al. (2014d) pokazali, że sieci neuronowe mogą nauczyć się wyprowadzać całe sekwencje znaków pobranych z obrazu, zamiast po prostu identyfikować poszczególne obiekty. Poprzednio powszechnie sądzono, że tego rodzaju uczenie się wymaga etykietowania poszczególnych elementów sekwencji (Gülçehre and Bengio 2013). Rekurencyjne sieci neuronowe, jak wspomniany wcześniej model sekwencyjny LSTM, są teraz używane do modelowania związków pomiędzy sekwencjami oraz innych sekwencji, o nieustalonych danych wejściowych. Uczenie się wedle zasady od sekwencji do sekwencji zapewne zrewolucjonizuje kolejne zastosowanie: tłumaczenia maszynowe (Bahdanau et al. 2015).

Trend rosnącej złożoności doprowadził w konsekwencji do wprowadzenia neuronowych maszyn Turinga (Graves et al. 2014a), które uczą się czytać z komórek pamięci i zapisują arbitralną zawartość do tych komórek. Takie sieci neuronowe mogą nauczyć się prostych programów, na przykładach pożądanego zachowania. Mogą nauczyć się sortować listy liczb, mając jako dane sekwencje danych pomieszanych i uporządkowanych. Ta technika samoprogramowania dopiero się rodzi, ale w przyszłości może być w zasadzie zastosowana do niemal każdego zadania.

Innym osiągnięciem deep learningu jest jego rozwinięcie na dziedzinę **wspomaganego uczenia się**. W kontekście wzmacnianego uczenia się autonomiczny agent musi nauczyć się wykonywać zadanie metodą prób i błędów, bez żadnej pomocy ze strony człowieka. DeepMind pokazało, że system

wzmacnianego uczenia się oparty na deep learningu potrafi nauczyć się grać w gry wideo na Atari, osiągając w wielu zadaniach wyniki na poziomie człowieka (Mnih et al. 2015). Deep learning znacząco poprawił też wydajność wzmacnianego uczenia się w dziedzinie robotyki (Finn et al. 2015).

Wiele z tych zastosowań deep learningu przynosi wysokie zyski. Jest on obecnie wykorzystywany przez czołowe firmy, w tym Google, Microsoft, Facebook, IBM, Baidu, Apple, Adobe, Netflix, NVIDIA i NEC.

Postępy w dziedzinie deep learningu zależą w dużym stopniu od postępów w dziedzinie infrastruktury programistycznej. Biblioteki programów, jak Theano (Bergstra et al. 2010, Bastien et al. 2012), PyLearn2 (Goodfellow et al. 2013c), Torch (Collobert et al. 2011b), DistBelief (Dean et al. 2012), Caffe (Jia 2013), MXNet (Chen et al. 2015) i TensorFlow (Abadi et al. 2015), zawsze wspomagały ważne projekty badawcze, a także produkty komercyjne.

Deep learning ma wkład także w inne dziedziny nauki. Wiele splotowych sieci rozpoznawania obrazów zapewnia modelowanie procesów widzenia, które mogą badać neuronaukowcy (DiCarlo 2013). Deep learning zapewnia także użyteczne narzędzia do przetwarzania ogromnej ilości danych, pozwalając na użyteczne przewidywania w wielu dziedzinach nauki. Z powodzeniem wykorzystano je do przewidywania, jaki wpływ będą mieć na siebie molekuły, co pomaga firmom farmaceutycznym na tworzenie nowych leków (Dahl et al. 2014), poszukiwanie cząstek atomowych (Baldi et al. 2014) i automatyzowanie analizy obrazów mikroskopowych w celu utworzenia trójwymiarowej mapy ludzkiego mózgu (Knowles-Barley et al. 2014). Spodziewamy się, że deep learning pojawi się w przyszłości w wielu kolejnych dziedzinach nauki.

Podsumowując, deep learning, rozwijany w ciągu kilku ostatnich dekad, stanowi podejście do systemów uczących się, które wpłynęło mocno na naszą wiedzę o mózgu człowieka, statystykę i matematykę stosowaną. W ostatnich latach deep learning znacznie zyskał na popularności i zastosowaniach, głównie ze względu na komputery o większej mocy, większe zbiory danych oraz techniki szkolenia głębszych sieci. Nadchodzące lata są pełne wyzwań i możliwości, aby dalej poprawiać deep learning, przekraczając nowe granice.

I

Podstawy matematyki stosowanej i systemów uczących się

Ta część książki stanowi wprowadzenie do podstawowych pojęć matematycznych potrzebnych do zrozumienia deep learningu. Zaczyna się od ogólnych pojęć matematyki stosowanej, które pozwolą zdefiniować funkcje wielu zmiennych, znaleźć maksima i minima tych funkcji oraz określić stopień przekonania. Następnie opisywane są podstawowe cele systemów uczących się. W tej części wskazuje się, jak osiągnąć te cele, opisując model reprezentujący określone przekonania, projektując funkcje kosztów, które sprawdzają, jak te przekonania korespondują z rzeczywistością, oraz używając algorytmu szkoleniowego pozwalającego zminimalizować funkcję kosztów. Ta podstawowa struktura stanowi podstawę wielu algorytmów stosowanych w systemach uczących się, w tym podejścia, które nie jest głębokie. W kolejnych częściach książki w ramach tej struktury rozwiniemy algorytmy deep learningu.

2

Algebra liniowa

Algebra liniowa to dziedzina matematyki, która jest szeroko stosowana w nauce i inżynierii. Jest to matematyka ciągła, a nie dyskretna, więc wielu naukowców zajmujących się informatyką niewiele o niej wie. Dobre zrozumienie algebry liniowej jest konieczne do zrozumienia i wykorzystania wielu algorytmów systemów uczących się, w tym algorytmów deep learningu. Dlatego wprowadzenie do deep learningu zostało poprzedzone skoncentrowaną prezentacją kluczowych pojęć algebry liniowej.

Osoby znające algebrę liniową mogą pominąć ten rozdział. Czytelnikom, którzy mają doświadczenie i znają pojęcia, ale potrzebują przypomnienia kluczowych wzorów, warto polecić podręcznik *The Matrix Cookbook* (Petersen and Pedersen 2006). Tym, którzy wcale nie znają algebry liniowej, ten rozdział wystarczy, aby przeczytać resztę książki, ale gorąco zaleca się sięgnięcie również po inne źródła z zakresu nauki algebry liniowej, jak Shilov (1977). W tym rozdziale pominięto wiele ważnych tematów algebry liniowej, które nie są konieczne do zrozumienia deep learningu.

2.1. Skalary, wektory, macierze i tensory

Nauka algebry liniowej obejmuje kilka typów obiektów matematycznych.

- **Skalary.** Skalar to jedna liczba, co odróżnia go od innych obiektów algebry liniowej, które są zwykle tablicami złożonymi z wielu liczb. Skalary zapisujemy tu kursywą. Zwykle nadaje się im nazwy rozpoczynające się małą literą. Gdy są wprowadzane, podajemy, jakiego typu są liczbą. Na przykład można powiedzieć: „Niech $\boldsymbol{s} \in \mathbb{R}$ stanowi nachylenie prostej", definiując skalar rzeczywisty, lub: „Niech $n \in \mathbb{N}$ będzie liczbą jednostek" – gdy definiujemy liczbę naturalną.

- **Wektory.** Wektor to tablica liczb. Liczby są ustawione po kolei. Można zidentyfikować kolejne liczby za pomocą indeksu wskazującego położenie. Zwykle nadaje się wektorom nazwy pisane małą literą, pisząc je pogrubioną czcionką, jak $\boldsymbol{x}$. Elementy wektora identyfikujemy przez nazwę pisaną kursywą z indeksem. Pierwszy element wektora $\boldsymbol{x}$ to x_1, drugi to x_2 i tak dalej. Musimy też określić, jakiego typu liczby będą przechowywane w wektorze. Jeśli każdy element należy do $\mathbb{R}$, a wektor ma n elementów, to leży on w zbiorze powstałym z iloczynu kartezjańskiego $\mathbb{R}n$ razy, oznaczonym jako $\mathbb{R}^n$. Gdy musimy szybko określić elementy wektora, piszemy je w postaci kolumny ujętej w nawiasy kwadratowe:

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}. \tag{2.1}$$

  Traktujemy wektory jako identyfikację punktów w przestrzeni, przy czym każdy podaje współrzędną na innej osi.

  Czasami trzeba indeksować zbiór elementów wektora. W takim przypadku definiuje się zbiór zawierający indeksy i zapisuje zbiór jako indeks dolny. Na przykład, aby uzyskać dostęp do x_1, x_3 i x_6, definiuje się zbiór $S = \{1, 3, 6\}$ i pisze $\boldsymbol{x}_{\boldsymbol{S}}$. Używa się znaku $-$, aby indeksować dopełnienie tego zbioru. Na przykład $\boldsymbol{x}_{-1}$ to wektor zawierający wszystkie elementy $\boldsymbol{x}$, poza elementem x_1, natomiast $\boldsymbol{x}_{-\boldsymbol{S}}$ to wektor zawierający elementy $\boldsymbol{x}$ poza x_1, x_3 i x_6.

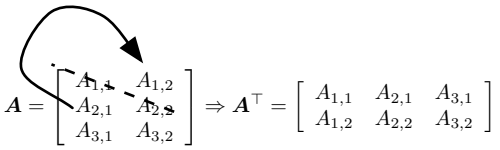
- **Macierze.** Macierz to dwuwymiarowa tablica liczb, więc każdy element jest identyfikowany przez dwa indeksy. Zwykle nazwy macierzy pisze się wielką literą z pogrubieniem, jak na przykład $\boldsymbol{A}$. Jeśli macierz rzeczywista $\boldsymbol{A}$ ma wysokość m i szerokość n, to mówimy, że $\boldsymbol{A} \in \mathbb{R}^{n \times m}$. Zwykle identyfikuje się elementy macierzy, pisząc jej nazwę kursywą, a nie czcionką pogrubioną, a indeksy oddziela przecinkami. Na przykład $A_{1,1}$ to lewy górny element $\boldsymbol{A}$, zaś $A_{m,n}$ to prawy dolny element. Wszystkie liczby o współrzędnej pionowej i zapisuje się jako ":" dla współrzędnej poziomej. Na przykład $A_{i,:}$ oznacza poprzeczny przekrój $\boldsymbol{A}$ ze współrzędną pionową i. Jest to określane jako i-ty wiersz macierzy $\boldsymbol{A}$. Podobnie $A_{:,i}$ to i-ta kolumna macierzy $\boldsymbol{A}$. Gdy chce się podać wszystkie elementy macierzy, zapisuje się je jako tablicę ujętą w nawiasy kwadratowe:

$$\begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix}. \tag{2.2}$$

Czasami trzeba indeksować wyrażenia macierzowe, gdzie nie pisze się pojedynczej litery. W takim przypadku używane są indeksy dolne po wyrażeniu, ale nie zmienia się liter na małe. Na przykład $f(\boldsymbol{A})_{i,j}$ daje element (i, j) macierzy obliczony po zastosowaniu na niej funkcji f.

- **Tensory.** W niektórych przypadkach potrzebne są tablice o większej liczbie wymiarów. W ogólnym przypadku liczby tablicy tworzą regularna siatkę ze zmienną liczbą osi, którą nazywamy tensorem. Oznaczamy tensor o nazwie „A" za pomocą innej czcionki: A. Element A o współrzędnych (i, j, k) zapisujemy jako $\mathsf{A}_{i,j,k}$.

Istotnym działaniem macierzowym jest **transpozycja**. Jest to lustrzane odbicie macierzy względem jej przekątnej, określanej jako **główna przekątna**, przebiegającej w dół i na prawo począwszy od lewego górnego narożnika. Na rysunku 2.1 widać graficzny obraz tego działania.



$$\boldsymbol{A} = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \\ A_{3,1} & A_{3,2} \end{bmatrix} \Rightarrow \boldsymbol{A}^\top = \begin{bmatrix} A_{1,1} & A_{2,1} & A_{3,1} \\ A_{1,2} & A_{2,2} & A_{3,2} \end{bmatrix}$$

Rysunek 2.1. Transpozycja macierzy to jej lustrzane odbicie wed[a]ug jej g[a]ównej przek¡tnej

Transpozycję macierzy $\boldsymbol{A}$ oznaczamy jako $\boldsymbol{A}^\top$ i zgodnie z definicją ma ona postać:

$$(\boldsymbol{A}^\top)_{i,j} = A_{j,i}. \tag{2.3}$$

Wektory można traktować jak macierze zawierające tylko jedną kolumnę. Transpozycja wektora przekształca go więc w macierz o jednym wierszu. Czasami definiuje się wektor, wypisując jego elementy w wierszu, jako macierz wierszową, a wtedy transpozycja przekształca ją na standardowy wektor kolumnowy, na przykład $\boldsymbol{x} = [x_1, x_2, x_3]^\top$.

Skalar można traktować jako macierz o jednym elemencie. Z tego wynika, że skalar stanowi swoją własną transpozycję: $a = a^\top$.

Macierze można do siebie dodawać, jeśli mają ten sam kształt, dodając ich odpowiadające sobie elementy: $\boldsymbol{C} = \boldsymbol{A} + \boldsymbol{B}$, gdzie $C_{i,j} = A_{i,j} + B_{i,j}$.

Można też dodać do macierzy skalar lub pomnożyć macierz przez skalar, wykonując odpowiednie działanie na każdym elemencie macierzy: $\boldsymbol{D} = a \cdot \boldsymbol{B} + c$, gdzie $D_{i,j} = aB_{i,j} + c$.

W kontekście deep learningu używa się także mniej konwencjonalnej notacji. Pozwalamy na dodanie macierzy i wektora, otrzymując nową macierz:

$\boldsymbol{C} = \boldsymbol{A} + \boldsymbol{b}$, gdzie $C_{i,j} = A_{i,j} + b_j$. Innymi słowy, wektor $\boldsymbol{b}$ zostaje dodany do każdego wiersza macierzy. Taki skrót eliminuje potrzebę definiowania macierzy, w której przed dodawaniem $\boldsymbol{b}$ jest kopiowany do każdego jej wiersza. Takie domyślne kopiowanie $\boldsymbol{b}$ do wielu lokalizacji nazywa się **rozgłaszaniem**.

2.2. Mnożenie macierzy i wektorów

Jednym z najważniejszych działań związanych z macierzami jest mnożenie dwóch macierzy. **Iloczynem macierzy** $\boldsymbol{A}$ i $\boldsymbol{B}$ jest trzecia macierz $\boldsymbol{C}$. Aby można było uzyskać iloczyn, macierz $\boldsymbol{A}$ musi mieć tyle samo kolumn, ile wierszy ma macierz $\boldsymbol{B}$. Jeśli $\boldsymbol{A}$ ma wymiary m na n, a $\boldsymbol{B}$ n na p, to $\boldsymbol{C}$ będzie miała rozmiar m na p. Iloczyn macierz możemy zapisać, umieszczając razem dwie macierze lub więcej, na przykład:

$$\boldsymbol{C} = \boldsymbol{AB}. \tag{2.4}$$

Iloczyn definiowany jest jako:

$$C_{i,j} = \sum_k A_{i,k} B_{k,j}. \tag{2.5}$$

Zauważmy, że standardowy iloczyn dwóch macierzy to nie jest po prostu macierz zawierająca iloczyny poszczególnych elementów. Takie działanie też istnieje i nosi nazwę **iloczynu po współrzędnych** lub **iloczynu Hadamarda** i jest oznaczane jako $\boldsymbol{A} \odot \boldsymbol{B}$.

Iloczyn skalarny (ang. *dot product*) między dwoma wektorami $\boldsymbol{x}$ i $\boldsymbol{y}$ o tych samych wymiarach jest iloczynem macierzy $\boldsymbol{x}^\top \boldsymbol{y}$. Możemy traktować iloczyn macierzy w postaci $\boldsymbol{C} = \boldsymbol{AB}$ jako wyznaczenie $C_{i,j}$ jako iloczynu skalarnego między wierszem macierzy $\boldsymbol{A}$ i kolumną macierzy $\boldsymbol{B}$.

Iloczyny macierzy mają wiele użytecznych własności, które sprawiają, że analiza matematyczna macierzy jest wygodniejsza. Na przykład mnożenie macierzy jest rozdzielne:

$$\boldsymbol{A}(\boldsymbol{B} + \boldsymbol{C}) = \boldsymbol{AB} + \boldsymbol{AC}. \tag{2.6}$$

Jest także łączne:

$$\boldsymbol{A}(\boldsymbol{BC}) = (\boldsymbol{AB})\boldsymbol{C}. \tag{2.7}$$

W przeciwieństwie do mnożenia skalarnego mnożenie macierzy nie jest przemienne (warunek $\boldsymbol{AB} = \boldsymbol{BA}$ nie zawsze jest spełniony). Jednak iloczyn z kropką dwóch wektorów jest przemienny:

$$\boldsymbol{x}^\top \boldsymbol{y} = \boldsymbol{y}^\top \boldsymbol{x}. \tag{2.8}$$

Transpozycja iloczynu macierzy ma prostą postać:

$$(\boldsymbol{AB})^\top = \boldsymbol{B}^\top \boldsymbol{A}^\top. \tag{2.9}$$

To pozwala nam na pokazanie równania 2.8, dzięki wykorzystaniu faktu, że wartość takiego iloczynu jest skalarem, a więc jest równa swojej transpozycji:

$$\boldsymbol{x}^\top \boldsymbol{y} = \left(\boldsymbol{x}^\top \boldsymbol{y}\right)^\top = \boldsymbol{y}^\top \boldsymbol{x}. \tag{2.10}$$

Ponieważ przedmiotem tej książki nie jest algebra liniowa, nie próbujemy tu podać wyczerpującej listy użytecznych własności iloczynu macierzy. Jednak czytelnicy powinni być świadomi, że istnieje o wiele więcej własności.

Wiemy już dość na temat zapisu algebry liniowej, aby zapisać układ równań liniowych:

$$\boldsymbol{Ax} = \boldsymbol{b}, \tag{2.11}$$

gdzie $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ to znana macierz, $\boldsymbol{b} \in \mathbb{R}^m$ to znany wektor, a $\boldsymbol{x} \in \mathbb{R}^n$ jest wektorem nieznanych zmiennych, które chcemy znaleźć. Każdy element x_i wektora $\boldsymbol{x}$ jest jedną z tych nieznanych zmiennych. Równanie 2.11 można zapisać jako:

$$\boldsymbol{A}_{1,:}\boldsymbol{x} = b_1 \tag{2.12}$$

$$\boldsymbol{A}_{2,:}\boldsymbol{x} = b_2 \tag{2.13}$$

$$\dots \tag{2.14}$$

$$\boldsymbol{A}_{m,:}\boldsymbol{x} = b_m \tag{2.15}$$

albo w bardziej jawnej postaci:

$$\boldsymbol{A}_{1,1}x_1 + \boldsymbol{A}_{1,2}x_2 + \cdots + \boldsymbol{A}_{1,n}x_n = b_1 \tag{2.16}$$

$$\boldsymbol{A}_{2,1}x_1 + \boldsymbol{A}_{2,2}x_2 + \cdots + \boldsymbol{A}_{2,n}x_n = b_2 \tag{2.17}$$

$$\dots \tag{2.18}$$

$$\boldsymbol{A}_{m,1}x_1 + \boldsymbol{A}_{m,2}x_2 + \cdots + \boldsymbol{A}_{m,n}x_n = b_m. \tag{2.19}$$

Notacja iloczynu macierzowo-wektorowa pozwala na bardziej zwięzłą reprezentację równań.

2.3. Macierze jednostkowe i odwrotne

Algebra liniowa oferuje potężne narzędzie nazywane **odwracaniem macierzy**, które pozwala analitycznie rozwiązać równanie 2.11 dla wielu wartości $\boldsymbol{A}$. Aby opisać odwracanie macierzy, musimy najpierw zdefiniować **macierze jednostkowe**. Macierz jednostkowa to macierz, która nie zmienia żadnego wektora, gdy mnożymy go przez tę macierz. Macierz tę, która zachowuje wartość wektorów n-wymiarowych, oznaczamy jako $\boldsymbol{I}_n$. Formalnie $\boldsymbol{I}_n \in \mathbb{R}$ oraz:

$$\forall \boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{I}_n \boldsymbol{x} = \boldsymbol{x}. \tag{2.20}$$

Struktura macierzy jednostkowej jest prosta: wszystkie elementy na głównej przekątnej mają wartość 1, a pozostałe mają wartość 0. Przykład pokazano na rysunku 2.2.

Macierz odwrotną do macierzy $\boldsymbol{A}$ oznaczamy jako $\boldsymbol{A}^{-1}$ i definiujemy jako:

$$\boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{I}_n. \tag{2.21}$$

Równanie 2.11 możemy rozwiązać w pokazanych niżej krokach:

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{2.22}$$

$$\boldsymbol{A}^{-1}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b} \tag{2.23}$$

$$\boldsymbol{I}_n\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b} \tag{2.24}$$

$$\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b}. \tag{2.25}$$

Oczywiście proces ten zależy od możliwości znalezienia $\boldsymbol{A}^{-1}$. Warunki istnienia macierzy odwrotnej opisujemy w kolejnym punkcie.

Gdy istnieje macierz odwrotna $\boldsymbol{A}^{-1}$, możemy ją wyznaczyć, stosując jeden z kilku istniejących algorytmów. W teorii ta sama macierz odwrotna może zostać użyta do wielokrotnego rozwiązania równania dla różnych wartości $\boldsymbol{b}$. Jednak $\boldsymbol{A}^{-1}$ jest użyteczna jako narzędzie teoretyczne i w praktyce nie powinna być używana w większości aplikacji programistycznych. Ponieważ macierz odwrotna może być wyznaczona na komputerach tylko z określoną dokładnością, algorytmy wykorzystujące wartość $\boldsymbol{b}$ pozwalają otrzymać bardziej dokładne estymację $\boldsymbol{x}$.

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rysunek 2.2. Przykład macierzy jednostkowej $\boldsymbol{I}_3$

2.4. Zależność liniowa i zakres

Aby istniała macierz $\boldsymbol{A}^{-1}$, równanie 2.11 musi mieć dokładnie jedno rozwiązanie dla każdej wartości $\boldsymbol{b}$. Możliwe jest, że układ równań nie ma żadnego rozwiązania lub ma nieskończenie wiele rozwiązań dla określonej wartości $\boldsymbol{b}$. Nie jest jednak możliwe, aby dla konkretnej wartości $\boldsymbol{b}$ było więcej niż jedno rozwiązanie, a mniej niż nieskończenie wiele. Jeśli zarówno $\boldsymbol{x}$, jak i $\boldsymbol{y}$ są rozwiązaniami, to:

$$\boldsymbol{z} = \alpha\boldsymbol{x} + (1 - \alpha)\boldsymbol{y} \tag{2.26}$$

jest także rozwiązaniem dla dowolnej wartości rzeczywistej α.

Aby stwierdzić, ile rozwiązań ma równanie, można potraktować kolumny $\boldsymbol{A}$ jako różne kierunki podróży z **początku** (punktu określonego przez wektor zawierający same zera), a potem określić, ile jest dróg prowadzących do $\boldsymbol{b}$. Z tego punktu widzenia każdy element $\boldsymbol{x}$ określa, jak daleko mamy podróżować w każdym z kierunków, a x_i określa, jak daleko mamy się poruszać w kierunku kolumny i:

$$\boldsymbol{A}\boldsymbol{x} = \sum_i x_i \boldsymbol{A}_{:,i}. \tag{2.27}$$

Taki rodzaj działania nosi nazwę **kombinacji liniowej**. Formalnie kombinacja liniowa pewnego zbioru wektorów $\{\boldsymbol{v}^{(1)}, \ldots, \boldsymbol{v}^{(n)}\}$ jest otrzymywana przez pomnożenie każdego wektora $\boldsymbol{v}^{(i)}$ przez odpowiadający mu współczynnik skalarny i dodanie do siebie wyników:

$$\sum_i c_i \boldsymbol{v}^{(i)}. \tag{2.28}$$

Zakres (ang. *span*) zbioru wektorów to zbiór wszystkich punktów, które można otrzymać na podstawie kombinacji liniowej oryginalnych wektorów.

Określenie, czy $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ ma rozwiązanie, sprowadza się więc do sprawdzenia, czy $\boldsymbol{b}$ znajduje się w zakresie kolumn $\boldsymbol{A}$. Zakres ten określamy jako **przestrzeń kolumny** lub **zasięg** $\boldsymbol{A}$.

Aby układ $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ miał rozwiązanie dla wszystkich wartości $\boldsymbol{b} \in \mathbb{R}^m$, musimy wymagać, aby przestrzeń kolumn $\boldsymbol{A}$ w całości należała do $\mathbb{R}^m$. Jeśli jakiś punkt w $\mathbb{R}^m$ jest poza przestrzenią kolumn, stanowi on potencjalnie wartość $\boldsymbol{b}$, dla której nie ma rozwiązania. Z wymagania, aby przestrzeń kolumn $\boldsymbol{A}$ należała w całości do $\mathbb{R}^m$, wynika, że $\boldsymbol{A}$ musi mieć co najmniej m kolumn, czyli $n \geq m$. W przeciwnym przypadku wymiarowość przestrzeni kolumn będzie mniejszy od m. Jako przykład rozpatrzmy przypadek macierzy 3 na 2. Docelowe $\boldsymbol{b}$ jest trójwymiarowe, lecz $\boldsymbol{x}$ jest tylko dwuwymiarowe, więc

modyfikowanie wartości $\boldsymbol{x}$ co najwyżej pozwala na określenie płaszczyzny dwuwymiarowej w obrębie $\mathbb{R}^3$. Równanie ma rozwiązanie wtedy i tylko wtedy, gdy $\boldsymbol{b}$ leży na tej płaszczyźnie.

Spełnienie $n \geq m$ jest jedynym koniecznym warunkiem istnienia rozwiązania. Nie jest to jednak warunek wystarczający, gdyż niektóre kolumny mogą być nadmiarowe. Rozważmy macierz 2 na 2, gdzie obie kolumny są identyczne. Jest to ta sama przestrzeń kolumn, jak dla macierzy 2 na 1, zawierającej jedna kopię powielonej kolumny. Innymi słowy, przestrzeń jest linią prostą i nie wskazuje wszystkich $\mathbb{R}^2$, mimo że ma dwie kolumny.

Formalnie ten rodzaj nadmiarowości jest znany jako **zależność liniowa**. Zbiór wektorów jest **liniowo niezależny**, jeśli żaden wektor ze zbioru nie jest kombinacją liniową pozostałych wektorów. Jeśli dodamy do zbioru wektor stanowiący kombinację liniową innych wektorów ze zbioru, to nowy wektor nie dodaje żadnych punktów do zakresu zbioru. Oznacza to, że aby przestrzeń kolumn macierzy obejmowała wszystkie $\mathbb{R}^m$, to macierz musi zawierać co najmniej jeden zbiór m liniowo niezależnych kolumn. Ten warunek jest konieczny i wystarczający do tego, aby równanie 2.11 miało rozwiązanie dla każdej wartości $\boldsymbol{b}$. Warto zauważyć, że zgodnie z wymaganiami zbiór musi mieć dokładnie m niezależnych kolumn, a nie co najmniej m. Żaden zbiór m-wymiarowych wektorów nie może mieć więcej niż m wzajemnie niezależnych liniowo kolumn, ale macierz, która ma więcej niż m kolumn, może mieć więcej niż jeden taki zbiór.

Aby macierz miała macierz odwrotną, musimy dodatkowo zapewnić, że równanie 2.11 ma nie więcej niż jedno rozwiązanie dla każdej wartości $\boldsymbol{b}$. Aby tego dokonać, musimy mieć pewność, że macierz ma nie więcej niż m kolumn. W przeciwnym przypadku jest więcej niż jeden sposób parametryzacji każdego rozwiązania. W sumie oznacza to, że macierz musi być **kwadratowa**, czyli $m = n$, a wszystkie kolumny muszą być liniowo niezależne. Macierz kwadratowa zawierająca liniowo zależne kolumny jest określana jako **osobliwa**. Jeśli $\boldsymbol{A}$ nie jest kwadratowa lub jest, ale jest osobliwa, to rozwiązanie równania jest nadal możliwe, ale nie można używać w tym celu macierzy odwrotnej.

Omówiliśmy dotąd odwrotność macierzy jako mnożoną lewostronnie. Można też zdefiniować odwrotność, która jest mnożona z prawej:

$$\boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{I}. \tag{2.29}$$

Dla macierzy kwadratowych odwrotności lewa i prawa są równe.

2.5. Normy

Czasami trzeba zmierzyć wielkość wektora. W systemach uczących się zwykle stosujemy w tym celu funkcję określaną jako **norma**. Formalnie norma L^p jest opisana wzorem:

$$||\boldsymbol{x}||_p = \Big(\sum_i |x_i|^p\Big)^{\frac{1}{p}} \tag{2.30}$$

dla $\boldsymbol{p} \in \mathbb{R}, \boldsymbol{p} \geq 1$.

Normy, w tym norma L^p, to funkcje odwzorowujące wektory na nieujemne wartości. Intuicyjnie norma wektora $\boldsymbol{x}$ mierzy odległość od początku układu współrzędnych do punktu $\boldsymbol{x}$. Ściślej norma to dowolna funkcja, która spełnia następujące cechy:

- $f(\boldsymbol{x}) = 0 \Rightarrow \boldsymbol{x} = 0$
- $f(\boldsymbol{x} + \boldsymbol{y}) \leq f(\boldsymbol{x}) + f(\boldsymbol{y})$ (**nierówność trójkąta**)
- $\forall \alpha \in \mathbb{R}, f(\alpha \boldsymbol{x}) = |\alpha| f(\boldsymbol{x})$

Norma L^2 przy $\boldsymbol{p} = 2$ znana jest jako **norma euklidesowa**, co oznacza odległość euklidesową od początku układu współrzędnych do punktu wskazanego przez $\boldsymbol{x}$. Norma L^2 jest często używana w systemach uczących się i często jest oznaczana $||\boldsymbol{x}||$, a indeks dolny 2 pomijamy. Często też używa się jej jako miary wielkości wektora za pomocą normy L^2 do kwadratu, obliczanej jako $\boldsymbol{x}^\top \boldsymbol{x}$.

Kwadratowa norma L^2 jest wygodniejsza matematycznie i obliczeniowo niż sama norma L^2. Na przykład każda pochodna kwadratowej normy L^2 względem każdego elementu $\boldsymbol{x}$ zależy tylko od odpowiedniego elementu $\boldsymbol{x}$, podczas gdy pochodne zwykłej normy L^2 zależą od całego wektora. W wielu kontekstach kwadratowa norma L^2 może być niepożądana, gdyż w pobliżu początku układu współrzędnych rośnie bardzo powoli. W niektórych zastosowaniach systemów uczących się ważne jest rozróżnienie elementów, które są dokładnie równe 0, od elementów małych, ale różnych od 0. W tych przypadkach sięga się po funkcję, która rośnie w takim samym stopniu w każdym miejscu, ale pozostaje prosta matematycznie: normy L^1. Norma L^1 w prostej postaci wygląda tak:

$$||\boldsymbol{x}||_1 = \sum_i |x_i|. \tag{2.31}$$

Norma L^1 jest powszechnie używana w systemach uczących się, gdy różnica między elementami o wartości zero i niezerowymi jest bardzo istotna. Za każdym razem, gdy element $\boldsymbol{x}$ oddala się od 0 o wartość ϵ, norma L^1 wzrasta o ϵ.

Czasami mierzymy wielkość wektora, licząc, ile jest w nim elementów różnych od 0. Niektórzy autorzy odwołują się do tej funkcji jako do „normy L^0", ale nie jest to poprawna terminologia. Liczba niezerowych elementów w wektorze nie jest normą, gdyż skalowanie wektora przez α nie zmienia liczby tych elementów. Norma L^1 jest często używana jako zastępstwo dla liczby elementów niezerowych.

Inną normą, która często pojawia się w systemach uczących się, jest L^∞, znana też jako **norma maksymalna**. Jest ona uproszczeniem i odwołuje się do wartości bezwzględnej największego co do modułu elementu wektora:

$$||\boldsymbol{x}||_\infty = \max_i |x_i|. \tag{2.32}$$

Czasami chcemy zmierzyć rozmiar macierzy. W kontekście deep learningu najpopularniejsze jest zastosowanie normy Frobeniusa, uważanej zwykle za przestarzałą:

$$||A||_F = \sqrt{\sum_{i,j} A_{i,j}^2}, \tag{2.33}$$

Stanowi ona analogię normy L^2 dla wektora.

Iloczyn z kropką dwóch wektorów można zapisać, uwzględniając normy:

$$\boldsymbol{x}^\top \boldsymbol{y} = ||\boldsymbol{x}||_2 ||\boldsymbol{y}||_2 \cos\theta, \tag{2.34}$$

gdzie θ jest kątem między $\boldsymbol{x}$ a $\boldsymbol{y}$.

2.6. Macierze i wektory specjalne

Niektóre specjalne rodzaje macierzy i wektorów są szczególnie przydatne. Macierze **diagonalne** zawierają głównie zera, a elementy niezerowe mają tylko na głównej przekątnej. Formalnie macierz $\boldsymbol{D}$ jest macierzą diagonalną wtedy i tylko wtedy, jeśli $D_{i,j} = 0$ dla każdego i $\neq$ j. Widzieliśmy już przykład takiej macierzy – macierz jednostkową, gdzie na głównej przekątnej są same jedynki. Kwadratową macierz diagonalną oznaczamy jako diag($\boldsymbol{v}$). Wartości na przekątnej podane są jako elementy wektora $\boldsymbol{v}$. Macierze diagonalne są w kręgu naszego zainteresowania, gdyż mnożenie przez nie jest wydajne obliczeniowo. Aby obliczyć diag($\boldsymbol{v}$)$\boldsymbol{x}$, trzeba tylko przeskalować każdy element przez v_i. Innymi słowy, diag($\boldsymbol{v}$)$\boldsymbol{x} = \boldsymbol{v} \odot \boldsymbol{x}$. Odwrócenie kwadratowej macierzy diagonalnej jest także wydajne. Macierz odwrotna istnieje tylko wtedy, gdy wszystkie elementy na głównej przekątnej są różne od 0. Wtedy diag($\boldsymbol{v}$)$^{-1}$ = = diag($[1/v_{1,...,}1/v_n]^\top$). W wielu przypadkach możemy wyprowadzić ogólny

algorytm systemów uczących się w postaci dowolnych macierzy, ale uzyskać mniej kosztowny (i prostszy w opisie) algorytm dzięki ograniczeniu części macierzy do macierzy diagonalnych.

Nie wszystkie macierze diagonalne muszą być kwadratowe. Możliwe jest zbudowanie prostokątnej macierzy diagonalnej. Nie można wprawdzie takiej macierzy odwrócić, ale można przez nią tanio mnożyć. Dla niekwadratowej macierzy $\boldsymbol{D}$ iloczyn $\boldsymbol{Dx}$ będzie wymagał skalowania każdego elementu $\boldsymbol{x}$ i scalania niektórych zer w wyniku, jeśli $\boldsymbol{D}$ jest wyższa niż szersza, albo odrzucenia niektórych końcowych elementów wektora, jeśli $\boldsymbol{D}$ jest szersza niż wyższa.

Macierz **symetryczna** to każda macierz, która równa się swojej transpozycji:

$$\boldsymbol{A} = \boldsymbol{A}^\top. \tag{2.35}$$

Macierze symetryczne pojawiają się, gdy elementy są wynikiem działania funkcji dwóch argumentów, przy czym wartość nie zależy od ich kolejności. Jeśli na przykład $\boldsymbol{A}$ jest macierzą pomiarów odległości, gdzie $\boldsymbol{A}_{i,j}$ daje odległość od punktu i do punktu j, to $\boldsymbol{A}_{i,j} = \boldsymbol{A}_{j,i}$, gdyż funkcje odległości są symetryczne.

Wektor jednostkowy to wektor o **normie jednostkowej**:

$$||\boldsymbol{x}||_2 = 1. \tag{2.36}$$

Wektory $\boldsymbol{x}$ i $\boldsymbol{y}$ są względem siebie **ortogonalne**, jeśli $\boldsymbol{x}^\top \boldsymbol{y} = 0$. Jeśli oba wektory mają normy różne od 0, to oznacza, że są względem siebie pod kątem 90 stopni. W $\mathbb{R}^n$ nie więcej niż n wektorów może być wzajemnie ortogonalnych z niezerową normą. Jeśli wektory są nie tylko ortogonalne, lecz także mają jednostkową normę, to nazywamy je **ortonormalnymi**.

Macierz ortogonalna to macierz kwadratowa, której wiersze są wzajemnie ortonormalne i kolumny są wzajemnie ortonormalne:

$$\boldsymbol{A}^\top \boldsymbol{A} = \boldsymbol{A}\boldsymbol{A}^\top = \boldsymbol{I}. \tag{2.37}$$

Stąd wynika, że:

$$\boldsymbol{A}^{-1} = \boldsymbol{A}^\top, \tag{2.38}$$

więc macierze ortogonalne są w zakresie naszego zainteresowania, gdyż ich odwrotność jest bardzo prosta do obliczenia. Warto zwrócić uwagę na definicję macierzy ortogonalnych. Wbrew intuicji ich wiersze są nie tylko ortogonalne, lecz także w pełni ortonormalne. Nie ma oddzielnej nazwy na macierz, której wiersze lub kolumny są ortogonalne, ale nie są ortonormalne.

2.7. Rozkład na wartości własne

Wiele obiektów matematycznych można lepiej zrozumieć, dzieląc je na składniki lub znajdując ich właściwości, które są uniwersalne, niezależne od tego, jaki wybierzemy sposób prezentacji. Na przykład liczby całkowite można rozłożyć na czynniki pierwsze. Sposób reprezentacji liczby 12 będzie się różnić w zależności od tego, czy zapiszemy ją w systemie dziesiętnym czy binarnym, ale zawsze prawdziwa będzie zależność $12 = 2 \times 2 \times 3$. Z tego sposobu prezentacji możemy wyprowadzić cenne właściwości, takie jak to, że 12 nie jest podzielne przez 5 oraz że każda liczba całkowita będąca wielokrotnością 12 będzie podzielna przez 3.

Podobnie jak dowiadujemy się czegoś o liczbie całkowitej, rozkładając ją na czynniki pierwsze, możemy rozłożyć macierze w taki sposób, żeby uzyskać informacje o ich właściwościach, które nie są oczywiste na podstawie samej prezentacji ich elementów. Jednym z szeroko używanych sposobów dekompozycji macierzy jest **rozkład na wartości własne**, gdzie rozkładamy macierz na zbiór wektorów własnych i wartości własnych. **Wektor własny** (ang. *eigenvector*) macierzy kwadratowej $\boldsymbol{A}$ to niezerowy wektor $\boldsymbol{v}$, którego pomnożenie przez a zmienia tylko skalę $\boldsymbol{v}$:

$$\boldsymbol{A}\boldsymbol{v} = \lambda \boldsymbol{v}. \tag{2.39}$$

Skalar λ to **wartość własna** odpowiadająca wektorowi własnemu (można też znaleźć **lewostronny wektor własny,** taki, że $\boldsymbol{v}^\top \boldsymbol{A} = \lambda \boldsymbol{v}^\top$, ale zwykle zajmujemy się prawostronnymi wektorami własnymi). Jeśli $\boldsymbol{v}$ jest wektorem własnym $\boldsymbol{A}$, to jest nim także każdy przeskalowany wektor $s\boldsymbol{v}$, dla każdego $s \in \mathbb{R}, s \neq 0$. Co więcej, $s\boldsymbol{v}$ ma tę samą wartość własną, dlatego zwykle szukamy tylko jednostkowych wektorów własnych.

Przypuśćmy, że macierz $\boldsymbol{A}$ ma n niezależnych liniowo wektorów własnych $\{\boldsymbol{v}^{(1)}, \ldots, \boldsymbol{v}^{(n)}\}$ z odpowiadającymi im wartościami własnymi $\{\lambda_1, \ldots, \lambda_n\}$. Możemy połączyć wszystkie wektory własne w macierz $\boldsymbol{V}$ z jednym wektorem własnym w każdej kolumnie: $\boldsymbol{V} = [\boldsymbol{v}^{(1)}, \ldots, \boldsymbol{v}^{(n)}]$. Podobnie możemy scalić wartości własne w postaci wektora $\lambda = [\lambda_1, \ldots, \lambda_n]^\top$. **Dekompozycja własna** macierzy $\boldsymbol{A}$ ma postać:

$$\boldsymbol{A} = \boldsymbol{V}\mathrm{diag}(\boldsymbol{\lambda})\boldsymbol{V}^{-1}. \tag{2.40}$$

Widzimy, że budowa macierzy z określonymi wartościami własnymi i wektorami własnymi pozwala nam na rozszerzenie przestrzeni w pożądanym kierunku. Czasem chcemy **rozłożyć** macierz na jej wartości i wektory własne. Pomaga to w analizie określonych właściwości macierzy, podobnie jak rozkład

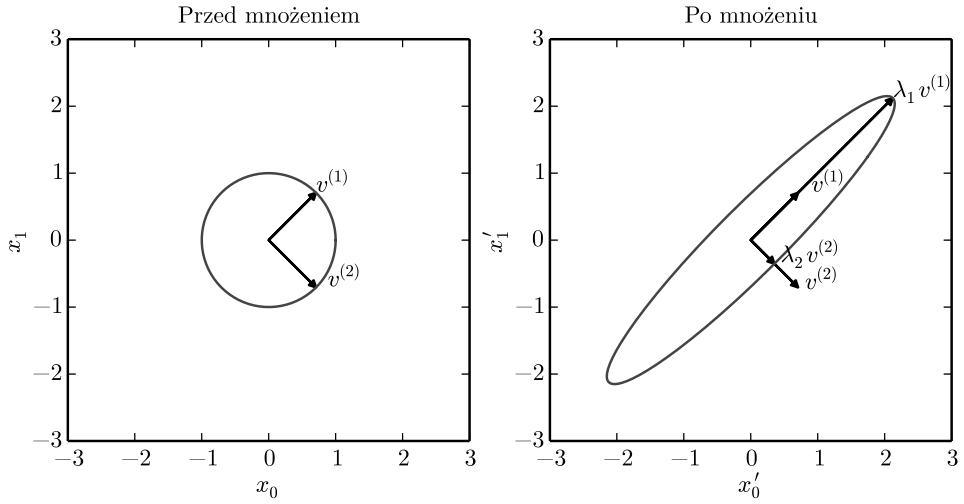
liczby całkowitej na czynniki pierwsze pozwala zrozumieć zachowanie się liczby całkowitej.

Nie każdą macierz można rozłożyć na wartości i wektory własne. W niektórych przypadkach dekompozycja jest możliwa, ale obejmuje liczby zespolone, a nie rzeczywiste. Na szczęście w tej książce musimy dokonywać rozkładu tylko określonego typu macierzy, które mają prosty rozkład. Konkretnie, każdą rzeczywistą macierz symetryczną można rozłożyć na wyrażenia, wykorzystując tylko rzeczywiste wartości i wektory własne:

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{\top}, \tag{2.41}$$

gdzie $\boldsymbol{Q}$ jest macierzą ortogonalną złożona z wektorów własnych $\boldsymbol{A}$, natomiast Λ jest macierzą diagonalną. Wartość własna $\Lambda_{i,i}$ jest powiązana z wektorem z kolumny i macierzy $\boldsymbol{Q}$ i oznaczona jako $Q_{:,i}$. Ponieważ macierz $\boldsymbol{Q}$ jest ortogonalna, można traktować $\boldsymbol{A}$ jako przestrzeń skalowania względem λ_i w kierunku $\boldsymbol{v}^{(i)}$. Przykład pokazano na rysunku 2.3.

Podczas gdy każdą rzeczywistą symetryczną macierz $\boldsymbol{A}$ na pewno można rozłożyć na elementy własne, dekompozycja ta może nie być unikatowa. Jeśli jeden lub więcej wektorów własnych ma tę samą wartość własną, to każdy zbiór ortogonalnych wektorów leżący w ich zakresie stanowi wektory własne



Rysunek 2.3. Efekt rozk[a]adu na warto±ci i wektory w[a]asne. Macierz$\boldsymbol{A}$ ma dwa ortogonalne wektory w[a]asne, gdzie$\boldsymbol{v}^{(1)}$ ma warto±¢ w[a]asn¡ λ_1, a $\boldsymbol{v}^{(2)}$ warto±¢ w[a]asn¡ λ_2. Po lewej: kre±limy zbiór wszystkich wektorów jednostkowych$\boldsymbol{u} \in \mathbb{R}^2$ jako okr¡g jednostkowy. Po prawej: kre±limy zbiór wszystkich punktów $\boldsymbol{Au}$. Obserwuj¡c sposób, w jaki $\boldsymbol{A}$ deformuje okr¡g jednostkowy, mo»na zobaczy¢, »e skaluje on przestrze« wag w kierunku$\boldsymbol{v}^{(i)}$ przez λ_i

z tą samą wartością własną i można wybrać macierz $\boldsymbol{Q}$, wykorzystującą te wektory własne zamiast innych. Przyjęto, że sortujemy elementy macierzy Λ w kolejności malejącej. Zgodnie z tą konwencją dekompozycja na elementy własne jest unikatowa tylko wtedy, gdy wartości własne są unikatowe.

Dekompozycja własna macierzy podaje wiele faktów o tej macierzy. Macierz jest osobliwa wtedy i tylko wtedy, gdy jedna z jej wartości własnych równa się 0. Dekompozycja własna rzeczywistej macierzy symetrycznej może być wykorzystana do optymalizacji wyrażeń kwadratowych w postaci $f(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x}$ względem $||\boldsymbol{x}||_2 = 1$. Gdy $\boldsymbol{x}$ jest równe wektorowi własnemu $\boldsymbol{A}$, to f przyjmuje wartość odpowiedniej wartości własnej. Maksymalna wartość f w ograniczonym obszarze stanowi maksymalną wartość własną, a wartość minimalna – minimalną wartość własną.

Macierz, której wszystkie wartości własne są dodatnie, jest nazywana **dodatnio określoną**. Jeśli wartości są większe lub równe 0, to macierz nazywamy **półokreśloną dodatnio**. Podobnie przy wartościach ujemnych macierz jest **ujemnie określona**, a przy wartościach własnych mniejszych lub równych 0 jest **półokreślona ujemnie**. Macierze półokreślone dodatnio są interesujące, gdyż gwarantują, że $\forall \boldsymbol{x}, \boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} \geq 0$. Macierze określone dodatnio dodatkowo zapewniają, że $\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} = 0 \Rightarrow \boldsymbol{x} = 0$.

2.8. Dekompozycja wartości osobliwej

W punkcie 2.7 opisano, jak dokonać dekompozycji macierzy na wartości i wektory własne. **Dekompozycja wartości osobliwej** (ang. *singular value decomposition*, SVD) daje jeszcze jeden sposób rozkładu macierzy na czynniki na **wektory osobliwe** i **wartości osobliwe**. SVD pozwala nam otrzymać te same informacje, które daje dekompozycja na czynniki własne; jednak SVD ma szersze zastosowanie. Każda macierz rzeczywista ma wartość osobliwą dekompozycji, co nie jest zawsze prawdą w odniesieniu do wartości własnych. Jeśli na przykład macierz nie jest kwadratowa, rozkład na czynniki własne nie jest zdefiniowany i musimy stosować dekompozycję na wartości osobliwe.

Przypomnijmy, że dekompozycja na wartości własne obejmuje analizę macierzy $\boldsymbol{A}$ w celu znalezienia macierzy $\boldsymbol{V}$ wektorów własnych i wektora wartości własnych $\boldsymbol{\lambda}$, dzięki którym możemy zapisać macierz $\boldsymbol{A}$ jako:

$$\boldsymbol{A} = \boldsymbol{V}\text{diag}(\boldsymbol{\lambda})\boldsymbol{V}^{-1}. \tag{2.42}$$

Dekompozycja na wartości osobliwe jest podobna, ale tym razem $\boldsymbol{A}$ zapisuje się jako iloczyn trzech macierzy:

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top. \tag{2.43}$$

Przypuśćmy, że $\boldsymbol{A}$ jest macierzą $m \times n$. Wtedy $\boldsymbol{U}$ jest macierzą $m \times m$, $\boldsymbol{D}$ macierzą $m \times n$, a $\boldsymbol{V}$ macierzą $n \times n$. Każda z tych macierzy ma zdefiniowaną specjalną strukturę. Macierze $\boldsymbol{U}$ i $\boldsymbol{V}$ są ortogonalne. Macierz $\boldsymbol{D}$ jest macierzą diagonalną. Zauważmy, że $\boldsymbol{D}$ nie musi być kwadratowa. Elementy na przekątnej macierzy $\boldsymbol{D}$ są **wartościami osobliwymi** macierzy $\boldsymbol{A}$. Kolumny $\boldsymbol{U}$ są **wektorami lewostronnie osobliwymi**, natomiast kolumny $\boldsymbol{V}$ są **wektorami prawostronnie osobliwymi**.

Dekompozycję macierzy $\boldsymbol{A}$ na wartości osobliwe interpretujemy jako dekompozycję na wartości własne funkcji $\boldsymbol{A}$. Wektory lewostronnie osobliwe macierzy $\boldsymbol{A}$ są wektorami osobliwymi $\boldsymbol{A}\boldsymbol{A}^\top$. Wektory prawostronnie osobliwe to wektory osobliwe $\boldsymbol{A}^\top\boldsymbol{A}$. Niezerowe wartości osobliwe macierzy $\boldsymbol{A}$ to pierwiastki kwadratowe wartości osobliwych $\boldsymbol{A}^\top\boldsymbol{A}$. To samo odnosi się do $\boldsymbol{A}\boldsymbol{A}^\top$.

Zapewne najbardziej użyteczną funkcją SVD jest to, że można jej użyć do częściowego uogólnienia odwracania niekwadratowych macierzy, co zostanie zaprezentowane w kolejnym punkcie.

2.9. Uogólniona macierz odwrotna (Moore'a–Penrose'a)

Odwracanie macierzy nie jest zdefiniowane dla macierzy niekwadratowych. Przypuśćmy, że chcemy dokonać lewostronnej odwrotności $\boldsymbol{B}$ macierzy $\boldsymbol{A}$, aby rozwiązać równanie liniowe:

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}, \tag{2.44}$$

mnożąc lewostronnie obie strony, aby uzyskać:

$$\boldsymbol{x} = \boldsymbol{B}\boldsymbol{y}. \tag{2.45}$$

Zależnie od struktury problemu jednoznaczne odwzorowanie z $\boldsymbol{A}$ na $\boldsymbol{B}$ może nie być możliwe. Jeśli macierz $\boldsymbol{A}$ jest wyższa niż szersza, równanie może nie mieć rozwiązania. Jeśli $\boldsymbol{A}$ jest szersza niż wyższa, to może być wiele możliwych rozwiązań.

Uogólniona macierz odwrotna (ang. *Moore-Penrose pseudoinverse*) pozwala na pewien postęp w tych przypadkach. Uogólniona odwrotność macierzy $\boldsymbol{A}$ zdefiniowana jest jako macierz:

$$\boldsymbol{A}^+ = \lim_{\alpha \searrow 0} (\boldsymbol{A}^\top\boldsymbol{A} + \alpha\boldsymbol{I})^{-1}\boldsymbol{A}^\top. \tag{2.46}$$

Praktycznie algorytm wyznaczenia uogólnionej macierzy odwrotnej nie opiera się na definicji, a na wzorze:

$$\boldsymbol{A}^{+} = \boldsymbol{V}\boldsymbol{D}^{+}\boldsymbol{U}^{\top}, \tag{2.47}$$

gdzie $\boldsymbol{U}, \boldsymbol{D}$ i $\boldsymbol{V}$ to wartości osobliwe dekompozycji $\boldsymbol{A}$, natomiast uogólnioną odwrotność $\boldsymbol{D}^{+}$ diagonalnej macierzy $\boldsymbol{D}$ otrzymujemy, biorąc odwrotność jej niezerowych elementów, a następnie transponując macierz wynikową.

Gdy $\boldsymbol{A}$ ma więcej kolumn niż wierszy, rozwiązanie równania liniowego za pomocą uogólnionej macierzy odwrotnej daje jedno z wielu możliwych rozwiązań. Konkretnie daje to spośród wielu rozwiązań $\boldsymbol{x} = \boldsymbol{A}^{+}\boldsymbol{y}$, które ma najmniejszą ze wszystkich normę euklidesową $||\boldsymbol{x}||_2$. Gdy $\boldsymbol{A}$ ma więcej wierszy niż kolumn, możliwe, że nie ma żadnego rozwiązania. W takim przypadku zastosowanie uogólnionej macierzy odwrotnej daje wartość $\boldsymbol{x}$, dla której $\boldsymbol{A}\boldsymbol{x}$ jest możliwie najbliższe $\boldsymbol{y}$ w sensie normy euklidesowej $||\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}||_2$.

2.10. Operator śladowy

Operator śladowy daje sumę wszystkich elementów na przekątnej macierzy:

$$\mathrm{Tr}(\boldsymbol{A}) = \sum_{i} \boldsymbol{A}_{i,i}. \tag{2.48}$$

Operator śladowy jest przydatny z wielu powodów. Niektóre działania, które są trudne do określenia bez sprowadzenia do operacji dodawania, można określić za pomocą iloczynu macierzy i operatora śladowego. Na przykład operator śladowy daje alternatywny sposób zapisu normy Frobeniusa dla macierzy:

$$||A||_F = \sqrt{\mathrm{Tr}(\boldsymbol{A}\boldsymbol{A}^{\top})}. \tag{2.49}$$

Pisanie wyrażenia w kontekście operatora śladowego otwiera możliwości manipulowania wyrażeniem za pomocą wielu użytecznych tożsamości. Na przykład operator śladowy jest niezmienny względem operatora transpozycji:

$$\mathrm{Tr}(\boldsymbol{A}) = \mathrm{Tr}(\boldsymbol{A}^{\top}). \tag{2.50}$$

Ślad macierzy kwadratowej składa się z wielu czynników i też jest niezmienny względem przeniesienia ostatniego czynnika na pierwszą pozycję, jeśli kształty odpowiednich macierzy pozwalają na zdefiniowanie powstałego iloczynu:

$$\mathrm{Tr}(\boldsymbol{A}\boldsymbol{B}\boldsymbol{C}) = \mathrm{Tr}(\boldsymbol{C}\boldsymbol{A}\boldsymbol{B}) = \mathrm{Tr}(\boldsymbol{B}\boldsymbol{C}\boldsymbol{A}) \tag{2.51}$$

lub bardziej ogólnie:

$$\mathrm{Tr}\Big(\prod_{i=1}^{n} \boldsymbol{F}^{(i)}\Big) = \mathrm{Tr}\Big(\boldsymbol{F}^{(n)} \prod_{i=1}^{n-1} \boldsymbol{F}^{(i)}\Big). \tag{2.52}$$

Ta niezmienność względem cyklicznych permutacji utrzymuje się nawet wtedy, gdy wynikowy iloczyn ma inny kształt. Na przykład dla $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ i $\boldsymbol{B} \in \mathbb{R}^{n\times m}$ mamy:

$$\mathrm{Tr}(\boldsymbol{AB}) = \mathrm{Tr}(\boldsymbol{BA}), \tag{2.53}$$

mimo że $\boldsymbol{AB} \in \mathbb{R}^{m\times m}$, a $\boldsymbol{BA} \in \mathbb{R}^{n\times n}$.

Innym pożytecznym faktem jest to, że skalar jest swoim własnym śladem: $\mathrm{a} = \mathrm{Tr}(\boldsymbol{a})$.

2.11. Wyznacznik

Wyznacznik macierzy kwadratowej, oznaczany jako $\det(\boldsymbol{A})$, to funkcja odwzorowująca macierz na skalary rzeczywiste. Wyznacznik jest równy iloczynowi wszystkich wartości własnych macierzy. Wartość bezwzględną wyznacznika można traktować jako miarę, na ile mnożenie przez macierz rozszerza lub zawęża przestrzeń. Jeśli wyznacznik jest równy 0, to przestrzeń zostaje całkowicie zmniejszona wzdłuż przynajmniej jednego wymiaru, co powoduje utratę objętości. Jeśli wyznacznik jest równy 1, to przekształcenie zachowuje objętość.

2.12. Przykład: analiza głównych składowych

Na podstawie samej znajomości algebry liniowej można wyprowadzić jeden prosty algorytm dla systemów uczących się. Jest to **analiza głównych składowych** (ang. *principal components analysis*, PCA).

Przypuśćmy, że mamy zbiór m punktów $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(n)}\}$ w $\mathbb{R}^n$ i chcemy zastosować ich stratną kompresję. Kompresja stratna oznacza zapisanie punktów w sposób, który wymaga mniej pamięci, ale powoduje utratę dokładności. Chcemy stracić jak najmniej na dokładności. Jednym ze sposobów na zakodowanie tych punktów jest reprezentowanie ich wersji w mniejszej liczbie wymiarów. Dla każdego punktu $\boldsymbol{x}^{(i)} \in \mathbb{R}^n$ znajdziemy odpowiadający mu wektor kodowania $\boldsymbol{c}^{(i)} \in \mathbb{R}^i$. Jeśli l jest mniejsze niż n, przechowywanie punktów kodu zajmie mniej pamięci, niż zapisanie przechowywanie oryginalnych danych. Będziemy chcieli znaleźć jakąś funkcję kodowania, która utworzy kod

dla danych wejściowych $f(\boldsymbol{x}) = \boldsymbol{c}$, oraz funkcję dekodowania, która utworzy zrekonstruowane dane wejściowe na podstawie ich kodu $\boldsymbol{x} \approx g(f(\boldsymbol{x}))$.

Wybieramy PCA jako naszą funkcję dekodowania. Aby dekoder był bardzo prosty, wybieramy mnożenie macierzy do odwzorowania kodu z powrotem na $\mathbb{R}^n$. Niech $g(\boldsymbol{c}) = \boldsymbol{D}\boldsymbol{c}$, gdzie $\boldsymbol{D} \in \mathbb{R}^{n \times l}$ jest macierzą definiującą dekodowanie. Wyznaczenie optymalnego kodu dla tego dekodera może stanowić problem. Aby zagadnienie kodowania było proste, PCA ogranicza kolumny macierzy $\boldsymbol{D}$ do takich, które są względem siebie ortogonalne (zauważmy, że macierz $\boldsymbol{D}$ nie jest formalnie macierzą ortogonalną, jeśli nie jest spełniony warunek $l = n$). Przy tak zdefiniowanym problemie możliwe jest wiele rozwiązań, gdyż możemy zwiększyć skalę macierzy $D_{:,i}$, jeśli proporcjonalnie zmniejszymy c_i dla wszystkich punktów. Aby problem miał jedno rozwiązanie, ograniczamy kolumny $\boldsymbol{D}$, tak aby miały jednostkową normę.

Aby przemienić ten pomysł w algorytm, który można wdrożyć, trzeba określić, jak wygenerować optymalny punkt kodu $\boldsymbol{c}^*$ dla każdego punktu wejściowego $\boldsymbol{x}$. Jednym ze sposobów jest zminimalizowanie odległości między punktem wejściowym $\boldsymbol{x}$ a jego rekonstrukcją $g(\boldsymbol{c}^*)$. Można zmierzyć tę odległość za pomocą normy. W algorytmie głównych składowych używamy normy L^2:

$$\boldsymbol{c}^* = \arg\min_{\boldsymbol{c}} ||\boldsymbol{x} - g(\boldsymbol{c})||_2. \tag{2.54}$$

Można przejść na normę kwadratową L^2 zamiast samej normy L^2, gdyż obie są minimalizowane przez tę samą wartość $\boldsymbol{c}$. Dzieje się tak dlatego, że norma L^2 jest nieujemna i podniesienie jej do kwadratu monotonicznie zwiększa nieujemne argumenty:

$$\boldsymbol{c}^* = \arg\min_{\boldsymbol{c}} ||\boldsymbol{x} - g(\boldsymbol{c})||_2^2. \tag{2.55}$$

Minimalizowana funkcja sprowadza się do:

$$(\boldsymbol{x} - g(\boldsymbol{c}))^\top (\boldsymbol{x} - g(\boldsymbol{c})) \tag{2.56}$$

(według definicji normy L^2 z równania 2.30)

$$= \boldsymbol{x}^\top \boldsymbol{x} - \boldsymbol{x}^\top g(\boldsymbol{c}) - g(\boldsymbol{c})^\top \boldsymbol{x} + g(\boldsymbol{c})^\top g(\boldsymbol{c}) \tag{2.57}$$

(zgodnie z własnością rozdzielności)

$$= \boldsymbol{x}^\top \boldsymbol{x} - 2\boldsymbol{x}^\top g(\boldsymbol{c}) + g(\boldsymbol{c})^\top g(\boldsymbol{c}) \tag{2.58}$$

(gdyż skalar $g(\boldsymbol{c})^\top \boldsymbol{x}$ jest równy swojej własnej transpozycji).

Teraz można ponownie zmienić minimalizowaną funkcję, aby ominąć pierwszy składnik, gdyż nie jest on zależny od $\boldsymbol{c}$:

$$\boldsymbol{c}^* = \arg\min_{\boldsymbol{c}} -2\boldsymbol{x}^\top g(\boldsymbol{c}) + g(\boldsymbol{c})^\top g(\boldsymbol{c}). \tag{2.59}$$

Dalej musimy zrobić podstawienie w definicji $g(\boldsymbol{c})$:

$$\boldsymbol{c}^* = \arg\min_{\boldsymbol{c}} -2\boldsymbol{x}^\top \boldsymbol{D}\boldsymbol{c} + \boldsymbol{c}^\top \boldsymbol{D}^\top \boldsymbol{D}\boldsymbol{c} \tag{2.60}$$

$$= \arg\min_{\boldsymbol{c}} -2\boldsymbol{x}^\top \boldsymbol{D}\boldsymbol{c} + \boldsymbol{c}^\top \boldsymbol{I}_l \boldsymbol{c} \tag{2.61}$$

(przez ortogonalność i ograniczenia normy jednostkowej $\boldsymbol{D}$)

$$= \arg\min_{\boldsymbol{c}} -2\boldsymbol{x}^\top \boldsymbol{D}\boldsymbol{c} + \boldsymbol{c}^\top \boldsymbol{c}. \tag{2.62}$$

Możemy rozwiązać ten problem optymalizacyjny za pomocą rachunku wektorowego (patrz punkt 4.3, jeśli nie wiesz, jak to się robi):

$$\nabla_{\boldsymbol{c}}(-2\boldsymbol{x}^\top \boldsymbol{D}\boldsymbol{c} + \boldsymbol{c}^\top \boldsymbol{c}) = \boldsymbol{0} \tag{2.63}$$

$$-2\boldsymbol{D}^\top \boldsymbol{x} + 2\boldsymbol{c} = \boldsymbol{0} \tag{2.64}$$

$$\boldsymbol{c} = \boldsymbol{D}^\top \boldsymbol{x}. \tag{2.65}$$

To sprawia, że algorytm jest wydajny: możemy optymalnie zakodować $\boldsymbol{x}$ za pomocą działania wektorowo macierzowego. Aby zakodować wektor, stosuje się funkcję kodera:

$$f(\boldsymbol{x}) = \boldsymbol{D}^\top \boldsymbol{x}. \tag{2.66}$$

Korzystając dalej z mnożenia macierzy, można także zdefiniować działanie rekonstrukcji PCA:

$$r(\boldsymbol{x}) = g\left(f\left(\boldsymbol{x}\right)\right) = \boldsymbol{D}\boldsymbol{D}^\top \boldsymbol{x}. \tag{2.67}$$

Dalej należy wybrać macierz kodowania $\boldsymbol{D}$. Aby to zrobić, wracamy do minimalizacji odległości L^2 między wejściami a rekonstrukcjami. Ponieważ używamy tej samej macierzy $\boldsymbol{D}$ do odkodowania wszystkich punktów, nie możemy dalej rozpatrywać tych punktów oddzielnie. Przeciwnie, musimy zminimalizować normę Frobeniusa macierzy błędów względem wszystkich wymiarów i punktów:

$$\boldsymbol{D}^* = \arg\min_{\boldsymbol{D}} \sqrt{\sum_{i,j} \left(x_j^{(i)} - r(\boldsymbol{x}^{(i)})_j\right)^2} \text{ dla } \boldsymbol{D}^\top \boldsymbol{D} = \boldsymbol{I}_l. \tag{2.68}$$

Aby wyprowadzić algorytm znajdowania $\boldsymbol{D}^+$, rozpoczynamy od rozważenia przypadku, gdzie $l = 1$. Wtedy $\boldsymbol{D}$ jest po prostu pojedynczym wektorem $\boldsymbol{d}$. Po podstawieniu równania 2.67 do równania 2.68 i uproszczeniu $\boldsymbol{D}$ na $\boldsymbol{d}$ problem sprowadza się do:

$$\boldsymbol{d}^* = \arg\min_{\boldsymbol{d}} \sum_i ||\boldsymbol{x}^{(i)} - \boldsymbol{d}\boldsymbol{d}^\top \boldsymbol{x}^{(i)}||_2^2 \text{ dla } ||\boldsymbol{d}||_2 = 1. \tag{2.69}$$

Powyższe sformułowanie jest najbardziej bezpośrednim sposobem wykonania podstawienia, ale nie jest to najlepszy sposób zapisu równania. Umieszcza się wartość skalarną $d_> x_{(i)}$ na prawo od wektora $\boldsymbol{d}$. Skalarne współczynniki są zwykle zapisywane po lewej stronie wektora, względem którego są stosowane. Dlatego zwykle zapisuje się ten wzór w postaci:

$$\boldsymbol{d}^* = \arg\min_{\boldsymbol{d}} \sum_i ||\boldsymbol{x}^{(i)} - \boldsymbol{d}^\top \boldsymbol{x}^{(i)} \boldsymbol{d}||_2^2 \text{ dla } ||\boldsymbol{d}||_2 = 1, \tag{2.70}$$

lub, wykorzystując fakt, że skalar jest swoją własną transpozycją, jako:

$$\boldsymbol{d}^* = \arg\min_{\boldsymbol{d}} \sum_i ||\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(i)\top} \boldsymbol{d}\boldsymbol{d}||_2^2 \text{ dla } ||\boldsymbol{d}||_2 = 1. \tag{2.71}$$

Czytelnicy powinni przyzwyczaić się do takich kosmetycznych przekształceń.

W tym miejscu pomocne może być przepisanie problemu według przykładów pojedynczego projektu macierzy, zamiast stosowania sumy oddzielnych przykładowych wektorów. To pozwoli na bardziej zwięzły zapis. Niech $\boldsymbol{X} \in \mathbb{R}^{m \times n}$ będzie macierzą zdefiniowaną przez ułożenie w stos wszystkich wektorów opisujących punkty $X_{i,:} = \boldsymbol{x}^{(i)T}$. Można teraz przepisać problem jako:

$$\boldsymbol{d}^* = \arg\min_{\boldsymbol{d}} ||\boldsymbol{X} - \boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top||_F^2 \text{ dla } \boldsymbol{d}^\top \boldsymbol{d} = 1. \tag{2.72}$$

Pomijając na chwilę ograniczenie, można uprościć część normy Frobeniusa w następujący sposób:

$$\arg\min_{\boldsymbol{d}} ||\boldsymbol{X} - \boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top||_F^2 \tag{2.73}$$

$$= \arg\min_{\boldsymbol{d}} \operatorname{Tr}\left(\left(\boldsymbol{X} - \boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top\right)^\top \left(\boldsymbol{X} - \boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top\right)\right) \tag{2.74}$$

(zgodnie z równaniem 2.49)

$$= \arg\min_{\boldsymbol{d}} \operatorname{Tr}(\boldsymbol{X}^\top \boldsymbol{X} - \boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top - \boldsymbol{d}\boldsymbol{d}^\top \boldsymbol{X}^\top \boldsymbol{X} + \boldsymbol{d}\boldsymbol{d}^\top \boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top) \tag{2.75}$$

$$= \arg\min_{\boldsymbol{d}} \operatorname{Tr}(\boldsymbol{X}^\top \boldsymbol{X}) - \operatorname{Tr}(\boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top) - \operatorname{Tr}(\boldsymbol{d}\boldsymbol{d}^\top \boldsymbol{X}^\top \boldsymbol{X}) + \operatorname{Tr}(\boldsymbol{d}\boldsymbol{d}^\top \boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top) \tag{2.76}$$

$$= \arg\min_{\boldsymbol{d}} - \operatorname{Tr}(\boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top) - \operatorname{Tr}(\boldsymbol{d}\boldsymbol{d}^\top \boldsymbol{X}^\top \boldsymbol{X}) + \operatorname{Tr}(\boldsymbol{d}\boldsymbol{d}^\top \boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top) \tag{2.77}$$

(ponieważ składniki niedotyczące $\boldsymbol{d}$ nie wpływają na $\arg\min$)

$$= \arg\min_{\boldsymbol{d}} -2\,\mathrm{Tr}(\boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top) + \mathrm{Tr}(\boldsymbol{d}\boldsymbol{d}^\top \boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top) \tag{2.78}$$

(ponieważ można zmieniać cyklicznie kolejność macierzy w śladzie, zgodnie z równaniem 2.52)

$$= \arg\min_{\boldsymbol{d}} -2\,\mathrm{Tr}(\boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top) + \mathrm{Tr}(\boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top \boldsymbol{d}\boldsymbol{d}^\top) \tag{2.79}$$

(używając ponownie tej samej własności).

W tym miejscu ponownie wprowadzamy ograniczenie: $\arg\min$:

$$\arg\min_{\boldsymbol{d}} -2\,\mathrm{Tr}(\boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top) + \mathrm{Tr}(\boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top \boldsymbol{d}\boldsymbol{d}^\top) \text{ dla } \boldsymbol{d}^\top \boldsymbol{d} = 1 \tag{2.80}$$

$$= \arg\min_{\boldsymbol{d}} -2\,\mathrm{Tr}(\boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top) + \mathrm{Tr}(\boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top) \text{ dla } \boldsymbol{d}^\top \boldsymbol{d} = 1 \tag{2.81}$$

(w związku z ograniczeniem)

$$= \arg\min_{\boldsymbol{d}} -\mathrm{Tr}(\boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top) \text{ dla } \boldsymbol{d}^\top \boldsymbol{d} = 1 \tag{2.82}$$

$$= \arg\max_{\boldsymbol{d}} \mathrm{Tr}(\boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{d}\boldsymbol{d}^\top) \text{ dla } \boldsymbol{d}^\top \boldsymbol{d} = 1 \tag{2.83}$$

$$= \arg\max_{\boldsymbol{d}} \mathrm{Tr}(\boldsymbol{d}^\top \boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{d}) \text{ dla } \boldsymbol{d}^\top \boldsymbol{d} = 1. \tag{2.84}$$

Ten problem optymalizacyjny może zostać rozwiązany za pomocą dekompozycji własnej. W szczególności optymalna wartość $\boldsymbol{d}$ jest podana przez wektor własny $\boldsymbol{X}^\top \boldsymbol{X}$ odpowiadający największej wartości własnej.

To wyprowadzenie jest właściwa dla przypadku $l = 1$ i podaje tylko pierwszą główną składową. Bardziej ogólnie, jeśli chcemy uzyskać bazę głównych składowych, macierz $\boldsymbol{D}$ jest określona przez l wektorów własnych odpowiadających największej wartości własnej. To można udowodnić przez indukcję. Zalecamy jako ćwiczenie przeprowadzenie takiego dowodu.

Algebra liniowa jest jedną z podstawowych dziedzin matematyki potrzebnych do zrozumienia deep learningu. Innym kluczowym obszarem matematyki niezbędnym w systemach uczących się jest teoria prawdopodobieństwa, prezentowana w kolejnym rozdziale.

3

Prawdopodobieństwo i teoria informacji

W tym rozdziale opisujemy teorię prawdopodobieństwa oraz teorię informacji. Teoria prawdopodobieństwa to matematyczny system reprezentacji niepewności. Daje środki pomiaru niepewności, a także aksjomaty wyprowadzania nowych stwierdzeń związanych z niepewnością. W zastosowaniach sztucznej inteligencji używa się teorii prawdopodobieństwa na dwa podstawowe sposoby. Po pierwsze prawa prawdopodobieństwa mówią, jak systemy AI powinny rozumować, aby można projektować algorytmy do wyznaczania lub aproksymacji wyrażeń wyprowadzonych za pomocą teorii prawdopodobieństwa. Po drugie możemy używać prawdopodobieństwa i statystyki do teoretycznej analizy zachowania proponowanych systemów AI.

Teoria prawdopodobieństwa to podstawowe narzędzie w wielu dziedzinach nauki i inżynierii. Ten rozdział służy temu, aby czytelnicy, którzy mają wykształcenie w dziedzinie programowania i mało poznali teorię prawdopodobieństwa, mogli zrozumieć zawartość tej książki.

Podczas gdy teoria prawdopodobieństwa pozwala tworzyć niepewne stwierdzenia i rozważać zagadnienia z niepewnością, teoria informacji pozwala nam na oszacowanie niepewności ilościowo jako rozkład prawdopodobieństwa.

Każdy, kto zna teorię prawdopodobieństwa oraz teorię informacji, może pominąć ten rozdział, poza punktem 3.14, w którym opisano grafy stosowane do opisu strukturalnych modeli probabilistycznych w systemach uczących się. Jeśli nie macie żadnego doświadczenia w tych dziedzinach, ten rozdział wystarczy, aby prowadzić projekty deep learningu, ale warto rozszerzyć swoją wiedzę z innych źródeł, jak na przykład Jaynes (2003).

3.1. Dlaczego prawdopodobieństwo?

Wiele gałęzi nauk komputerowych ma do czynienia z elementami, które są całkowicie deterministyczne i pewne. Programista może zwykle bezpiecznie założyć, że procesor wykona każdą instrukcję bez błędów. Błędy sprzętowe się zdarzają, ale są na tyle rzadkie, że większość aplikacji programistycznych nie musi ich brać pod uwagę. W związku z tym, że wielu informatyków i programistów działa we względnie czystym i pewnym środowisku, zaskoczeniem może być dla nich szerokie zastosowanie teorii prawdopodobieństwa w deep learningu.

Systemy uczące się mają do czynienia z niepewnymi ilościami, a czasami też z elementami stochastycznymi (niedeterministycznymi). Niepewność i stochastyka mogą pochodzić z wielu źródeł. Począwszy od lat osiemdziesiątych XX wieku naukowcy przedstawili istotne argumenty na ocenę niepewności za pomocą prawdopodobieństwa. Wiele przedstawianych tu argumentów stanowi podsumowanie lub jest zainspirowanych pracami Pearla (1988).

Prawie wszystkie działania wymagają wnioskowania w warunkach niepewności. W zasadzie poza twierdzeniami matematycznymi, które są z definicji prawdziwe, trudno znaleźć twierdzenia, które są z pewnością prawdziwe, lub zdarzenia, które na pewno będą miały miejsce. Oto trzy możliwe źródła niepewności:

1. Stochastyczna natura modelowanego systemu. Na przykład większość interpretacji w mechanice kwantowej opisuje dynamikę cząsteczek subatomowych jako probabilistyczne. Możemy też stworzyć teoretyczne scenariusze, które mają dynamikę losową, jak hipotetyczna gra w karty, gdzie zakładamy, że karty są naprawdę potasowane w kolejności losowej.

2. Niepełne możliwości obserwacji. Nawet systemy deterministyczne mogą okazać się stochastyczne, gdy nie można obserwować wszystkich zmiennych, które wpływają na zachowanie systemu. Na przykład w paradoksie Monty'ego Halla* uczestnicy gry mogą wybrać jedno z trojga drzwi, a wygrana znajduje się za zamkniętymi drzwiami. Dwoje drzwi prowadzi do kozy, a trzecie do samochodu. Wynik wyboru uczestnika gry jest deterministyczny, ale z jego punktu widzenia jest on losowy.

3. Niekompletne modelowanie. Gdy korzysta się z modelu, w którym trzeba odrzucić część zaobserwowanych informacji, to odrzucone informacje wprowadzają niepewność do przewidywań modelu. Przypuśćmy, że budujemy robota, który potrafi dokładnie określić położenie każdego

*Od nazwiska prezentera teleturnieju popularnego w USA i Kanadzie.

otaczającego go obiektu. Jeśli robot podczas przewidywania przyszłego położenia przetwarza przestrzeń na wartości dyskretne, to zabieg ten sprawia, że nie potrafi on precyzyjnie przewidzieć położenia obiektów: każdy obiekt może być w dowolnym miejscu dyskretnej komórki, w której ma się znajdować.

W wielu przypadkach bardziej praktyczne jest skorzystanie z prostej, lecz niepewnej zasady niż z pewnej, ale skomplikowanej, jeśli nawet zasada jest deterministyczna i system modelowania ma możliwość spełnienia skomplikowanej zasady. Na przykład prosta zasada: „Większość ptaków lata" jest prostsza do wprowadzenia i szerzej użyteczna niż zasada w postaci: „Ptaki latają, poza bardzo młodymi, które się jeszcze nie nauczyły latać, chorymi lub skaleczonymi, które utraciły możliwość latania, niektórymi gatunkami ptaków nielotnych, w tym kazuarów, strusi, kiwi...", która jest trudna do zastosowania, utrzymania i rozpowszechniania, a po pokonaniu trudności jest nadal niepewna i narażona na niepowodzenie.

Podczas gdy powinno być jasne, że musimy mieć środki do prezentowania i rozważania niepewności, nie jest oczywiste, że teoria prawdopodobieństwa może dać wszystkie narzędzia potrzebne w zastosowaniach sztucznej inteligencji. Teoria prawdopodobieństwa powstała w celu analizy częstości występowania zdarzeń. Łatwo zobaczyć, jak stosować ją do analizy zdarzeń, takich jak wylosowanie określonych kart w rozdaniu podczas gry w pokera. Tego rodzaju zdarzenia są często powtarzalne. Gdy mówimy, że wynik uzyskamy z prawdopodobieństwem p, to znaczy, że jeśli powtórzymy eksperyment (np. rozdawanie kart) nieskończenie wiele razy, to w wynikach nastąpi p powtórzeń. Ten sposób rozumowania nie wydaje się możliwy do natychmiastowego zastosowania do propozycji, które się nie powtarzają. Jeśli lekarz zbada pacjenta i powie, że ma on na 40% grypę, to znaczy coś całkiem innego – nie możemy mieć nieskończenie wielu kopii pacjenta ani nie ma powodu wierzyć, że kopie pacjenta miałyby te same objawy przy różnych warunkach początkowych. W przypadku diagnozy lekarskiej pacjenta korzystamy z prawdopodobieństwa, aby pokazać **stopień przekonania**, gdzie 1 oznacza absolutną pewność, że pacjent ma grypę, a 0 to absolutna pewność, że grypy nie ma. Pierwszy rodzaj prawdopodobieństwa, powiązany bezpośrednio z częstością występowania zdarzeń, znany jest jako **prawdopodobieństwo częstotliwościowe**, natomiast ten drugi, który wiąże się z jakościowymi poziomami pewności, znany jest jako **prawdopodobieństwo bayesowskie**.

Jeśli wymienimy listę kilku właściwości zdroworozsądkowego rozumowania o niepewności, to jedynym sposobem, aby je spełnić, jest traktowanie prawdopodobieństwa bayesowskiego jako zachowującego się identycznie jak prawdopodobieństwo częstotliwościowe. Jeśli na przykład chcemy wyznaczyć

prawdopodobieństwo wygranej w pokera, zakładając, że gracz ma określony zestaw kart, używamy dokładnie tych samych wzorów, jak w przypadku wyznaczania prawdopodobieństwa, iż pacjent ma daną chorobę, pod warunkiem że ma określone objawy. Więcej przykładów tego, że mały zbiór zdroworozsądkowych założeń pozwala stosować te same aksjomaty do obu rodzajów prawdopodobieństwa, można znaleźć w książce Ramseya (1926).

Można patrzeć na prawdopodobieństwo jak na logiczny układ z niepewnością. Logika podaje zbiór formalnych reguł do określenia, które propozycje mogą być prawdziwe, a które fałszywe. Teoria prawdopodobieństwa daje zbiór formalnych reguł do określenia wiarygodności tezy, że propozycja jest prawdziwa, z uwagi na prawdopodobieństwo innych propozycji.

3.2. Zmienne losowe

Zmienna losowa to zmienna, która może losowo przyjmować różne wartości. Zwykle oznaczana jest małą literą pisaną zwykłą czcionką, a przyjmowane przez nią wartości przyjmują postać indeksów dolnych. Na przykład x_1 i x_2 to dwie możliwe wartości zmiennej losowej x. Dla zmiennych wektorowych zapiszemy zmienną losową jako $\mathbf{x}$, a jedną z jego wartości jako $\boldsymbol{x}$. Pojedyncza zmienna losowa opisuje po prostu możliwe stany; musi być powiązana z rozkładem prawdopodobieństwa, który określa, jaka jest możliwość wystąpienia każdego z tych stanów.

Zmienne losowe mogą być dyskretne lub ciągłe. Dyskretna zmienna losowa to taka, która ma skończoną lub też nieskończoną, ale policzalną liczbę stanów. Warto zwrócić uwagę, że stany te nie muszą być wartościami całkowitymi; mogą też być nazwane, więc wcale nie muszą mieć wartości liczbowej. Ciągła zmienna losowa jest powiązana z wartością rzeczywistą.

3.3. Rozkłady prawdopodobieństwa

Rozkład prawdopodobieństwa to opis, jaka jest możliwość ustawienia zmiennej losowej lub wielu zmiennych losowych na każdą możliwą wartość. Sposób opisu rozkładu zależy od tego, czy zmienne są dyskretne, czy ciągłe.

3.3.1. Zmienne dyskretne i funkcje masy prawdopodobieństwa

Rozkład prawdopodobieństwa względem zmiennych dyskretnych można opisać za pomocą **funkcji masy prawdopodobieństwa** (PMF). Zwykle oznaczamy funkcje masy prawdopodobieństwa wielką literą P. Często łączymy

każdą zmienną losową z różną funkcją masy prawdopodobieństwa i czytelnik musi wywnioskować, której PMF użyć na podstawie tożsamości zmiennej losowej, a nie na podstawie nazwy funkcji; $P(\mathrm{x})$ jest zwykle różna od $P(\mathrm{y})$.

Funkcja masy prawdopodobieństwa odwzorowuje stan zmiennej losowej na prawdopodobieństwo przyjęcia przez nią danego stanu. Prawdopodobieństwo, że $\mathrm{x} = x$, jest oznaczone jako $P(\mathrm{x})$, gdzie prawdopodobieństwo 1 wskazuje, że równość $\mathrm{x} = x$ jest pewna, natomiast 0 wskazuje, że równość $\mathrm{x} = x$ jest niemożliwa*. Czasami, aby ujednoznacznić, którą PMF zastosować, piszemy nazwę zmiennej losowej jawnie $P(\mathrm{x} = x)$. Czasami najpierw definiujemy zmienną, a następnie używamy zapisu $\sim$, aby określić, jaki ma ona rozkład: $\mathrm{x} \sim P(\mathrm{x})$.

Funkcja masy prawdopodobieństwa może w tym samym czasie działać na wielu zmiennych. Taki rozkład prawdopodobieństwa znany jest jako **wspólny rozkład prawdopodobieństwa**. $P(\mathrm{x} = x, \mathrm{y} = y)$ wskazuje prawdopodobieństwo, że jednocześnie $\mathrm{x} = x$ i $\mathrm{y} = y$. Zapis możemy skrócić do postaci $P(\mathrm{x}, \mathrm{y})$.

Aby funkcja P była PMF względem zmiennej x, musi mieć następujące właściwości:

- dziedzina P musi być zbiorem wszystkich możliwych stanów x
- $\forall x \in \mathrm{x}, 0 \leq P(\mathrm{x}) \leq 1$ – zdarzenie niemożliwe ma prawdopodobieństwo równe 0 i żaden stan nie może mieć mniejszego prawdopodobieństwa; podobnie zdarzenie z pewnością się wydarzy, jeśli ma prawdopodobieństwo równe 1, a żaden stan nie może mieć większej szansy zdarzenia się
- $\sum_{x \in \mathrm{x}} P(\mathrm{x}) = 1$ – tę właściwość traktujemy jako **znormalizowaną;** bez niej można by otrzymać prawdopodobieństwa większe od 1, jeśli obliczanoby prawdopodobieństwo zajścia jednego z wielu zdarzeń.

Rozważmy na przykład pojedynczą dyskretną zmienną losową x, która może mieć k stanów. Można wprowadzić **rozkład jednostajny** x – czyli każdy stan będzie jednakowo prawdopodobny – nadając jego funkcji PMF wartość:

$$P(\mathrm{x} = x_i) = \frac{1}{k} \tag{3.1}$$

dla każdego i. Możemy zobaczyć, że wszystkie wymagania funkcji masy prawdopodobieństwa są spełnione. Wartość $\frac{1}{k}$ jest dodatnia, gdyż k jest liczbą całkowitą dodatnią. Widzimy także, że:

$$\sum_i P(\mathrm{x} = x_i) = \sum_i \frac{1}{k} = \frac{k}{k} = 1, \tag{3.2}$$

więc rozkład jest poprawnie znormalizowany.

*Zwykle mówimy „bardzo mało prawdopodobne", gdyż najczęściej wartość 0 jest wartością graniczną.

3.3.2. Zmienne ciągłe i funkcje gęstości prawdopodobieństwa

Podczas pracy z ciągłymi zmiennymi losowymi opisujemy rozkład prawdopodobieństwa za pomocą **funkcji gęstości prawdopodobieństwa** (PDF), a nie za pomocą funkcji masy prawdopodobieństwa. Aby funkcja p była funkcją gęstości prawdopodobieństwa, musi mieć następujące właściwości:

- dziedziną p musi być zbiór wszystkich możliwych stanów x
- $\forall x \in \mathrm{x}, p(x) \geq 0$ – zauważmy, że nie ma wymogu, aby $p(x) \leq 1$
- $\int p(x)dx = 1$.

Funkcja gęstości prawdopodobieństwa $p(x)$ nie daje bezpośrednio prawdopodobieństwa określonego stanu. Zamiast tego podaje prawdopodobieństwo znalezienia się wewnątrz nieskończenie małego obszaru, którego objętość δx podana jest jako $p(x)\delta x$.

Można scałkować funkcję gęstości, aby znaleźć masę prawdopodobieństwa dla zbioru punktów. W szczególności prawdopodobieństwo, że x leży w pewnym zbiorze $\mathbb{S}$, jest podane jako całka z $p(x)$ względem tego zbioru. W przykładzie jednowymiarowym prawdopodobieństwo, że x należy do przedziału $[a, b]$ jest określone jako $\int_{[\mathrm{a,b}]} p(x)dx$.

Dla funkcji PDF odpowiadającej określonej gęstości prawdopodobieństwa względem ciągłej zmiennej losowej rozpatrzmy rozkład jednostajny przedziału liczb rzeczywistych. Można to zrobić za pomocą funkcji $u(x; a, b)$, gdzie a i b to końce przedziału, przy czym $b > a$. Notacja z przecinkiem oznacza „względem parametru"; rozpatrujemy x jako argument funkcji, podczas gdy a i b są parametrami definiującymi funkcję. Aby sprawić, że żadna masa prawdopodobieństwa nie znajduje się poza przedziałem, mówimy, że $u(x; a, b) = 0$ dla każdego $x \notin [a, b]$. Wewnątrz $[a; b]$, $u(x; a, \mathrm{b} = \frac{1}{b-a}$. Jest ona wszędzie nie mniejsza od 0, ponadto całka wynosi 1. Często oznaczamy, że x ma rozkład jednostajny w przedziale $[a, b]$, zapisując to jako $\mathrm{x} \sim U(a, b)$.

3.4. Prawdopodobieństwo brzegowe

Czasami znamy rozkład prawdopodobieństwa względem zbioru zmiennych i chcemy poznać rozkład prawdopodobieństwa dla ich podzbioru. Rozkład prawdopodobieństwa dla podzbioru określany jest jako **brzegowy rozkład prawdopodobieństwa**. Przypuśćmy na przykład, że mamy dyskretne zmienne losowe x i y oraz znamy $P(\mathrm{x}, \mathrm{y})$. Możemy znaleźć $P(\mathrm{x})$ wg **reguły sum**:

$$\forall x \in \mathrm{x}, P(\mathrm{x} = x) = \sum_{y} P(\mathrm{x} = x, \mathrm{y} = y). \tag{3.3}$$

Nazwa „prawdopodobieństwa brzegowe" pochodzi od procesu jego obliczana na papierze. Gdy wartości $P(\mathrm{x}, \mathrm{y})$ zapisujemy na siatce z różnymi wartościami x w wierszach i y w kolumnach, w naturalny sposób sumujemy je wzdłuż wiersza siatki, a potem zapisujemy $P(x)$ na brzegu kartki papieru na prawo od wiersza. Dla zmiennych ciągłych musimy korzystać z całkowania w miejsce sumowania:

$$p(x) = \int p(x, y)dy. \tag{3.4}$$

3.5. Prawdopodobieństwo warunkowe

W wielu przypadkach interesuje nas prawdopodobieństwo jakiegoś zdarzenia, pod warunkiem że zaszło już inne zdarzenie. Określamy je jako **prawdopodobieństwo warunkowe**. Oznaczamy je tak, że $\mathrm{y} = y$, wiedząc, że $\mathrm{x} = x$ jako $P(\mathrm{y} = y \mid \mathrm{x} = x)$. Takie prawdopodobieństwo warunkowe możemy wyznaczyć wg wzoru:

$$P(\mathrm{y} = y \mid \mathrm{x} = x) = \frac{P(\mathrm{y} = y, \mathrm{x} = x)}{P(\mathrm{x} = x)}. \tag{3.5}$$

Prawdopodobieństwo warunkowe jest zdefiniowane tylko wtedy, gdy $P(\mathrm{x} = = x) > 0$; nie możemy obliczyć prawdopodobieństwa warunkowego uwarunkowanego zdarzeniem, które nigdy nie zajdzie.

Ważne jest, aby nie pomylić prawdopodobieństwa warunkowego z wyznaczeniem tego, co się zdarzy, jeśli podejmiemy jakieś działanie. Prawdopodobieństwo warunkowe, że osoba pochodzi z Niemiec, jeżeli mówi po niemiecku, jest dość wysokie, ale jeśli losowo wybrana osoba zostanie nauczona mówić po niemiecku, jej kraj pochodzenia się nie zmieni. Wyznaczenie konsekwencji działania określa się jako wykonanie **interwencyjnego zapytania**. Zapytania takie stanowią domenę **modelowania przyczynowego** i nie są przedmiotem tej książki.

3.6. Reguła łańcuchowa w prawdopodobieństwie warunkowym

Każdy wspólny rozkład prawdopodobieństwa dla wielu zmiennych losowych może zostać rozłożony na rozkłady warunkowe względem tylko jednej zmiennej:

$$P(\mathrm{x}^{(1)}, \ldots, \mathrm{x}^{(n)}) = P(\mathrm{x}^{(1)}) \Pi_{i=2}^{n} P(\mathrm{x}^{(i)} \mid \mathrm{x}^{(1)}, \ldots, \mathrm{x}^{(i-1)}). \tag{3.6}$$

Nosi to nazwę **reguły łańcuchowej** lub **reguły iloczynowej prawdopodobieństwa** i wynika bezpośrednio z prawdopodobieństwa warunkowego zawartego we wzorze 3.5. Na przykład dwukrotne zastosowanie tej definicji da poniższy wynik:

$$\begin{aligned} P(\mathrm{a},\mathrm{b},\mathrm{c}) &= P(\mathrm{a} \mid \mathrm{b},\mathrm{c})P(\mathrm{b},\mathrm{c}) \\ P(\mathrm{b},\mathrm{c}) &= P(\mathrm{b} \mid \mathrm{c})P(\mathrm{c}) \\ P(\mathrm{a},\mathrm{b},\mathrm{c}) &= P(\mathrm{a} \mid \mathrm{b},\mathrm{c})P(\mathrm{b} \mid \mathrm{c})P(\mathrm{c}) \end{aligned}$$

3.7. Niezależność oraz niezależność warunkowa

Dwie zmienne losowe x i y są **niezależne**, jeśli ich rozkład prawdopodobieństwa można wyrazić jako iloczyn dwóch czynników, gdzie jeden dotyczy tylko x, a drugi tylko y:

$$\forall x \in \mathrm{x}, y \in \mathrm{y},\ p(\mathrm{x}=x, \mathrm{y}=y) = p(\mathrm{x}=x)p(\mathrm{y}=y). \tag{3.7}$$

Dwie zmienne losowe są **warunkowo niezależne** względem zmiennej losowej z, jeśli prawdopodobieństwo warunkowe względem x i y rozkłada się na czynniki dla każdej wartości z wg zależności:

$$\forall x \in \mathrm{x}, y \in \mathrm{y}, z \in \mathrm{z},\ p(\mathrm{x}=x, \mathrm{y}=y \mid \mathrm{z}=z) = p(\mathrm{x}=x \mid \mathrm{z}=z)p(\mathrm{y}=y \mid \mathrm{z}=z). \tag{3.8}$$

Możemy oznaczyć niezależność i warunkową niezależność za pomocą zwartego zapisu $\mathrm{x} \perp \mathrm{y}$, co oznacza, że x i y są niezależne, natomiast $\mathrm{x} \perp \mathrm{y} \mid \mathrm{z}$ oznacza, że x i y są warunkowo niezależne dla z.

3.8. Wartość oczekiwana, wariancja i kowariancja

Wartość oczekiwana pewnej funkcji $f(x)$ względem rozkładu prawdopodobieństwa $P(\mathrm{x})$ to wartość średnia, którą f przyjmuje, gdy x zostanie wylosowane z P. Dla zmiennych dyskretnych można ją wyznaczyć jako sumę:

$$\mathbb{E}_{\mathrm{x} \sim P}[f(x)] = \sum_x P(x)f(x), \tag{3.9}$$

natomiast dla zmiennych ciągłych obliczamy ją za pomocą całki:

$$\mathbb{E}_{\mathrm{x} \sim p}[f(x)] = \int p(x)f(x)dx. \tag{3.10}$$

Gdy tożsamość rozkładu wynika jasno z kontekstu, można po prostu napisać nazwę zmiennej losowej, względem której liczymy wartość oczekiwaną, jako $\mathbb{E}_{\mathrm{x}}[f(x)]$. Jeśli jest jasne, której zmiennej losowej dotyczy wartość oczekiwana, można całkowicie pominąć indeks dolny jak w wyrażeniu $\mathbb{E}[f(x)]$. Domyślnie przyjmuje się, że $\mathbb{E}[\cdot]$ jest średnią względem wartości wszystkich zmiennych losowych wewnątrz nawiasu. Podobnie gdy nie ma niejednoznaczności, można pominąć nawiasy kwadratowe.

Wartości oczekiwane są liniowe, jak na przykład:

$$\mathbb{E}_{\mathrm{x}}[\alpha f(x) + \beta g(x)] = \alpha \mathbb{E}_{\mathrm{x}}[f(x)] + \beta \mathbb{E}_{\mathrm{x}}[g(x)], \tag{3.11}$$

gdy α i β nie są zależne od x.

Wariancja określa miarę tego, jak wartości zmiennej losowej x będą się różnić przy próbkowaniu różnych wartości x z jej rozkładu prawdopodobieństwa:

$$\mathrm{Var}(f(x)) = \mathbb{E}\left[(f(x) - \mathbb{E}[f(x)])^2\right]. \tag{3.12}$$

Gdy wariancja jest niska, wartości $f(x)$ skupiają się w pobliżu wartości oczekiwanej. Pierwiastek kwadratowy z wariancji określany jest jako **odchylenie standardowe**.

Kowariancja daje nam miarę tego, na ile dwie wartości są liniowo ze sobą powiązane, a także skalę tych zmiennych:

$$\mathrm{Cov}(f(x), g(y)) = \mathbb{E}\left[(f(x) - \mathbb{E}\left[f(x)\right])\left(g(y) - \mathbb{E}\left[g(y)\right]\right)\right]. \tag{3.13}$$

Duże wartości bezwzględne kowariancji oznaczają, że wartości ulegają dużym zmianom i są jednocześnie daleko od swoich odpowiednich średnich. Jeśli znak kowariancji jest dodatni, to obie zmienne będą miały jednocześnie względnie duże wartości. Jeśli znak kowariancji jest ujemny, to jedna zmienna ma relatywnie duże wartości, a jednocześnie druga zmienna relatywnie małe. Inne miary, jak **korelacja,** normalizują wkład każdej zmiennej, pozwalając tylko stwierdzić, w jakim stopniu zmienne są ze sobą powiązane, nie ma natomiast na nie wpływu wielkość pojedynczych zmiennych.

Pojęcia kowariancji i zależności są ze sobą związane, ale należy je traktować odrębnie. Są powiązane, gdyż dwie zmienne niezależne mają kowariancję równą 0, a zmienne o niezerowej kowariancji są zależne od siebie. Jednak niezależność to co innego niż kowariancja. Aby dwie zmienne miały zerową kowariancję, nie może być między nimi zależności liniowej. Niezależność jest mocniejszym wymaganiem niż zerowa wartość kowariancji, gdyż niezależność wyklucza także związki nieliniowe. Możliwe jest, aby dwie zmienne były zależne, ale miały zerową kowariancję. Przypuśćmy na przykład, że pobieramy pierwszą próbkę liczby rzeczywistej x z rozkładem jednostajnym w przedziale

$[-1, 1]$. Następna próbka to zmienna losowa s. Z prawdopodobieństwem $\frac{1}{2}$ określamy, że wartość s wynosi 1. W przeciwnym przypadku określamy, że wartością s jest -1. Możemy wygenerować zmienną losową y, przypisując jej $y = sx$. Jasno widać, że x i y nie są niezależne, gdyż x w pełni określa wielkość y. Jednak $\text{Cov}(x, y) = 0$.

Macierz kowariancji losowego wektora $x \in \mathbb{R}^n$ jest macierzą $n \times n$, jak na przykład:

$$\text{Cov}(\mathbf{x})_{i,j} = \text{Cov}(\text{x}_i, \text{x}_j). \tag{3.14}$$

Elementy diagonalne kowariancji dają wariancję:

$$\text{Cov}(\text{x}_i, \text{x}_i) = \text{Var}(\text{x}_i). \tag{3.15}$$

3.9. Znane rozkłady prawdopodobieństwa

Poniżej kilka prostych rozkładów prawdopodobieństwa, przydatnych w kontekście systemów uczących się.

3.9.1. Rozkład Bernoulliego

Rozkład Bernoulliego to rozkład jednej binarnej zmiennej losowej. Jest zależny od jednego parametru $\phi \in [0, 1]$, który daje prawdopodobieństwo, że zmienna losowa przyjmie wartość 1. Rozkład ma następujące cechy:

$$P(\text{x} = 1) = \phi \tag{3.16}$$

$$P(\text{x} = 0) = 1 - \phi \tag{3.17}$$

$$P(\text{x} = x) = \phi^x (1 - \phi)^{1-x} \tag{3.18}$$

$$\mathbb{E}_{\text{x}}[\text{x}] = \phi \tag{3.19}$$

$$\text{Var}_{\text{x}}(\text{x}) = \phi(1 - \phi) \tag{3.20}$$

3.9.2. Rozkład wielopunktowy

Rozkład wielopunktowy (inaczej kategoryczny) jest rozkładem jednej zmiennej dyskretnej o k różnych stanach, gdzie k ma wartość skończoną[1]. Rozkład

[1] „Rozkład wielopunktowy" to termin ukuty przez Gustava Lacerdo i spopularyzowany przez Murphy'ego (2012). Rozkład wielopunktowy jest specjalnym przypadkiem rozkładu wielomianowego. Rozkład wielomianowy to rozkład względem wektorów w $\{0, \ldots, n\}^k$, reprezentującej, ile razy każda z kategorii k jest odwiedzana, gdy pobierzemy n próbek z rozkładu wielopunktowego. W wielu tekstach termin „wielomianowy" oznacza rozkład wielopunktowy bez określenia, że odnosi się tylko do przypadku $n = 1$.

wielopunktowy jest parametryzowany przez wektor $p \in [0,1]^{k-1}$, gdzie p_i podaje prawdopodobieństwo i-tego stanu. Końcowe prawdopodobieństwo stanu k-tego jest określone jako $1 - \mathbf{1}^\top \boldsymbol{p}$. Zwróćmy uwagę, że musimy ograniczyć $\mathbf{1}^\top \boldsymbol{p} \leq 1$. Rozkłady wielopunktowe są często używane w celu odwołania się do rozkładu względem kategorii obiektów, więc zwykle nie zakładamy, że stan 1 ma wartość liczbową 1 i tak dalej. Dlatego nie musimy zwykle wyznaczać wartości oczekiwanej ani wariancji zmiennych losowych o rozkładzie wielopunktowym.

Rozkłady Bernoulliego i wielopunktowy wystarczają do opisania dowolnego rozkładu w ich dziedzinie. Mogą opisać każdy rozkład, ale nie dlatego, że mają szczególnie duże możliwości, lecz dlatego, że mamy do czynienia z prostą dziedziną: modelują zmienne dyskretne, w których można policzyć wszystkie możliwe stany. Gdy mamy do czynienia ze zmiennymi ciągłymi, istnieje niepoliczalna liczba stanów, więc każdy rozkład opisywany przez niewielką liczbę parametrów musi nakładać ścisłe ograniczenia na rozkład.

3.9.3. Rozkład Gaussa

Najczęściej używanym rozkładem względem liczb rzeczywistych jest **rozkład normalny**, znany jako **rozkład Gaussa**:

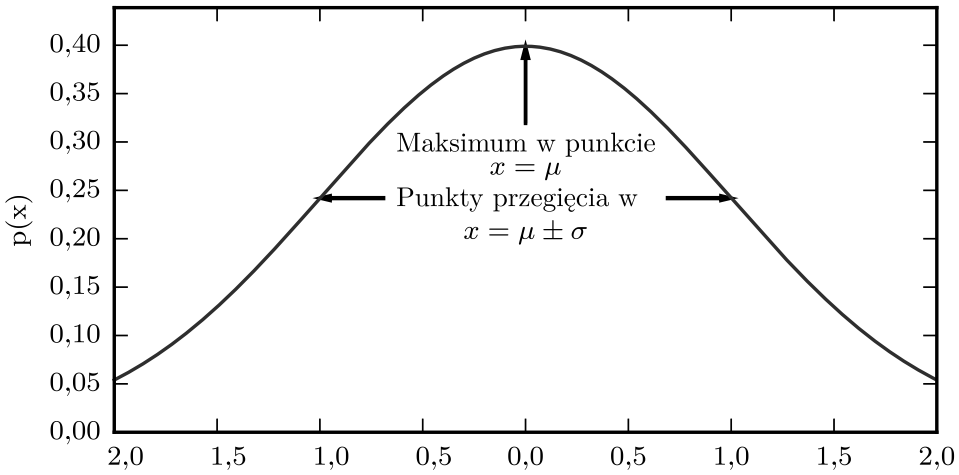
$$\mathcal{N}(x; \mu, \sigma^2) = \sqrt{\frac{1}{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right). \tag{3.21}$$

Na rysunku 3.1 pokazano wykres funkcji gęstości rozkładu normalnego.

Dwa parametry $\mu \in \mathbb{R}$ i $\sigma \in (0, \infty)$ sterują rozkładem normalnym. Parametr μ daje współrzędną punktu szczytowego. Jest to również średnia arytmetyczna rozkładu: $\mathbb{E}[\mathrm{x}] = \mu$. Odchylenie standardowe rozkładu jest oznaczone jako σ, a wariancja jako σ^2. Gdy wyznaczamy wartość PDF, musimy podnieść do kwadratu i odwrócić σ. Gdy musimy często wyznaczać wartość PDF dla różnych wartości parametru, bardziej efektywnym sposobem parametryzacji rozkładu jest użycie parametru $\beta \in (0, \infty)$ do kontroli **precyzji** lub odwrotnej wariancji rozkładu:

$$\mathcal{N}(x; \mu, \beta^{-1}) = \sqrt{\frac{\beta}{2\pi}} \exp\left(-\frac{1}{2}\beta(x-\mu)^2\right). \tag{3.22}$$

Rozkłady normalne są dobrym wyborem w wielu zastosowaniach. Gdy nie znamy formy rozkładu względem liczb rzeczywistych, rozkład normalny to dobry wybór domyślny z dwóch podstawowych powodów. Po pierwsze wiele rozkładów, które chcemy modelować, ma rozkład naprawdę bliski rozkładu



Rysunek 3.1. Rozkªad normalny N(x; μ, σ^2) ma postaĆ klasycznej krzywej dzwonowej , gdzie wspóªrz¦dna x szczytu jest okre±lona przez μ, a szeroko±Ć krzywej jest dana przez σ. W tym przykªadzie pokazujemy standardowy rozkªad normalny z warto±ciami $\mu = 0$ i $\sigma = 1$

normalnego. **Centralne twierdzenie graniczne** pokazuje, że suma wielu niezależnych zmiennych losowych ma średni rozkład normalny. W praktyce oznacza to, że wiele skomplikowanych systemów można z powodzeniem modelować jako szum o rozkładzie normalnym, jeśli nawet można je dekomponować na części o bardziej strukturalnym zachowaniu. Po drugie spośród wszystkich możliwych rozkładów prawdopodobieństwa o tej samej wariancji rozkład normalny koduje najwięcej niepewności względem liczb rzeczywistych, Możemy więc traktować rozkład normalny jako ten, który wykorzystuje w modelu najmniej wstępnej wiedzy. Pełne rozwinięcie i uzasadnienie tej tezy wymaga więcej narzędzi matematycznych i wrócimy do niego w punkcie 19.4.2.

Rozkład normalny jest uogólniony jako $\mathbb{R}^n$, co w tym przypadku określamy jako **rozkład normalny jednoczynnikowy**. Można go sparametryzować za pomocą symetrycznej macierzy dodatnio określonej $\boldsymbol{\Sigma}$:

$$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sqrt{\frac{1}{(2\pi)^n \det(\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu})\right). \qquad (3.23)$$

Parametr $\boldsymbol{\mu}$ nadal podaje średnią arytmetyczną rozkładu, ale teraz ma wartość wektora. Parametr $\boldsymbol{\Sigma}$ podaje macierzy kowariancję rozkładu. Podobnie jak w przypadku jednoczynnikowym, gdy chcemy wyznaczyć PDF kilka razy dla różnych wartości parametrów, kowariancja nie jest efektywnym obliczeniowo sposobem parametryzacji rozkładu, gdyż w celu obliczenia PDF

musimy odwrócić $\boldsymbol{\Sigma}$. Zamiast tego możemy użyć **macierzy precyzji** $\boldsymbol{\beta}$:

$$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\beta}^{-1}) = \sqrt{\frac{\det(\boldsymbol{\beta})}{(2\pi)^n}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\top \boldsymbol{\beta}(\boldsymbol{x}-\boldsymbol{\mu})\right). \tag{3.24}$$

Często ustala się macierz kowariancji jako macierz diagonalną. Jeszcze prostszy jest **równokierunkowy** rozkład Gaussa, którego kowariancja jest skalarem pomnożonym przez macierz jednostkową.

3.9.4. Rozkłady wykładniczy i Laplace'a

W kontekście deep learningu często chcemy uzyskać rozkład prawdopodobieństwa z ostrym punktem dla $x = 0$. Aby to osiągnąć, możemy wykorzystać **rozkład wykładniczy**:

$$p(x; \lambda) = \lambda \mathbf{1}_{x \geq 0} \exp(-\lambda x). \tag{3.25}$$

Rozkład wykładniczy wykorzystuje funkcję wskaźnikową $\mathbf{1}_{x \geq 0}$, aby przypisać zerowe prawdopodobieństwo wszystkim ujemnym wartościom x.

Blisko spokrewniony z nim rozkład prawdopodobieństwa, który pozwala umieścić ostry szczyt masy prawdopodobieństwa w arbitralnie wybranym punkcie μ, to **rozkład Laplace'a**:

$$\text{Laplace}(x; \mu, \gamma) = \frac{1}{2\gamma} \exp\left(-\frac{|x-\mu|}{\gamma}\right). \tag{3.26}$$

3.9.5. Rozkład Diraca i rozkład empiryczny

W niektórych przypadkach chcemy określić, że cała masa rozkładu prawdopodobieństwa skupia się wokół jednego punktu. Można to osiągnąć, definiując PDF za pomocą **delty Diraca** $\delta(x)$:

$$p(x) = \delta(x - \mu). \tag{3.27}$$

Funkcja delta Diraca jest zdefiniowana w ten sposób, że ma wartość 0 wszędzie poza zerem, a jej całka wynosi 1. Funkcja ta nie jest zwykła funkcją, która łączy każdą wartość x z rzeczywistym wynikiem. Jest innym rodzajem obiektu matematycznego określanym jako **funkcja uogólniona** zdefiniowana w sensie jej właściwości podczas całkowania. Możemy traktować deltę Diraca jako punkt graniczny ciągu funkcji, które przypisują coraz mniej masy wszystkim punktom poza 0.

Przez zdefiniowanie $p(x)$ jako wartości δ przesuniętej o $-\mu$ otrzymujemy nieskończenie wąski szczyt masy prawdopodobieństwa, gdzie $x = \mu$.

Powszechnym zastosowaniem rozkładu delta Diraca jest użycie go jako składnika **rozkładu empirycznego:**

$$\hat{p}(\boldsymbol{x}) = \frac{1}{m}\sum_{i=1}^{m}\delta(\boldsymbol{x} - \boldsymbol{x}^{(i)}). \tag{3.28}$$

Przypisuje on masę prawdopodobieństwa $\frac{1}{m}$ do każdego z m punktów $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}$ na podstawie zbioru danych lub kolekcji próbek. Rozkład delta Diraca jest konieczny jedynie do zdefiniowania rozkładu empirycznego względem zmiennych ciągłych. Dla zmiennych dyskretnych sytuacja jest prostsza: rozkład empiryczny można określić jako rozkład wielopunktowy z prawdopodobieństwem związanym z każdą możliwą wartością wejściową, która jest równa **empirycznej częstotliwości** tej wartości w zbiorze szkoleniowym.

Można traktować rozkład empiryczny utworzony ze zbioru danych przykładów testowych jako określenie rozkładu próbkowanego podczas szkolenia modelu na tym zbiorze danych. Inna ważną perspektywą dla rozkładu empirycznego jest fakt, że jest to gęstość prawdopodobieństwa maksymalizująca wiarygodność danych szkoleniowych (patrz punkt 5.5).

3.9.6. Rozkłady mieszane

Często definiuje się rozkłady prawdopodobieństwa, łącząc ze sobą prostsze rozkłady. Jednym ze sposobów łączenia rozkładów jest budowa **rozkładu mieszanego**. Składa się on z kilku rozkładów składowych. Przy każdej próbie wybór, którego składnika użyć do wygenerowania próbki, jest określony przez próbkowanie tożsamości składnika z rozkładu wielopunktowego:

$$P(\mathrm{x}) = \sum_{i} P(\mathrm{c} = i)P(\mathrm{x} \mid \mathrm{c} = i), \tag{3.29}$$

gdzie $P(\mathrm{c})$ to rozkład wielopunktowy względem tożsamości składników.

Widzieliśmy już jeden przykład rozkładu mieszanego – rozkład empiryczny względem wartości rzeczywistych to rozkład mieszany z jednym składnikiem Diraca dla każdego przykładu szkoleniowego.

Model mieszany jest prostą strategią łączenia rozkładów prawdopodobieństwa, aby utworzyć rozkład bogatszy. W rozdziale 16 bardziej szczegółowo analizujemy sztukę budowania złożonych rozkładów prawdopodobieństwa na podstawie tych prostych.

Model mieszany pozwala pobieżnie zapoznać się z pojęciem, które będzie później miało ogromne znaczenie – **zmiennej utajonej.** Zmienna utajona to zmienna losowa, której nie można bezpośrednio obserwować. Składowa

zmienna tożsamościowa c w modelu mieszanym posłuży jako przykład. Utajone zmienne mogą odnosić się do x za pośrednictwem wspólnego rozkładu, w tym przypadku $P(\mathrm{x}, \mathrm{c}) = P(\mathrm{x}|\mathrm{c})P(\mathrm{c})$. Rozkład $P(\mathrm{c})$ względem zmiennej utajonej oraz rozkład $P(\mathrm{x}|\mathrm{c})$ wiążący zmienne utajone ze zmiennymi widocznymi określa kształt rozkładu $P(\mathrm{x})$, mimo że można opisać $P(\mathrm{x})$ bez odwoływania się do zmiennej utajonej. Zmienne utajone są szerzej omawiane w punkcie 16.5.

Często stosowanym modelem mieszanym o dużych możliwościach jest **model mieszany Gaussa**, w którym składniki $p(\mathrm{x} \mid \mathrm{c} = \mathrm{i})$ są gaussowskie. Każdy składnik ma oddzielnie parametryzowaną średnią arytmetyczną $\boldsymbol{\mu}^{(i)}$ oraz kowariancję $\boldsymbol{\Sigma}^{(i)}$. Niektóre mieszanki mogą mieć więcej ograniczeń, na przykład określa się, że kowariancje powinny być wspólne dla składników przez ograniczenie $\boldsymbol{\Sigma}^{(i)} = \boldsymbol{\Sigma}, \forall i$. Podobnie jak w pojedynczym rozkładzie Gaussa, mieszanka rozkładów gaussowskich może ograniczać macierz kowariancji, wymuszając diagonalność lub izotropowość dla każdego ze swoich składników.

Poza średnimi arytmetycznymi i kowariancjami parametry mieszanki gaussowskiej określają **rozkład aprioryczny** (*a priori*) $\alpha_i = P(\mathrm{c} = i)$ dla każdego składnika i. Określenie *a priori* wskazuje, że wyrażone są oczekiwania modelu dotyczące c *zanim* nastąpi obserwacja $\mathbf{x}$. Analogicznie $P(\mathrm{c} \mid \boldsymbol{x})$ to prawdopodobieństw *a posteriori*, gdyż jest wyznaczane *po* obserwacji $\mathbf{x}$. Model mieszany Gaussa to **metoda uniwersalnej aproksymacji** gęstości w tym sensie, że każda gładka gęstość może zostać aproksymowana z określonym niezerowym błędem przez mieszany model Gaussa z dostateczną liczbą składników. Na rysunku 3.2 pokazano próbki z mieszanego modelu Gaussa.

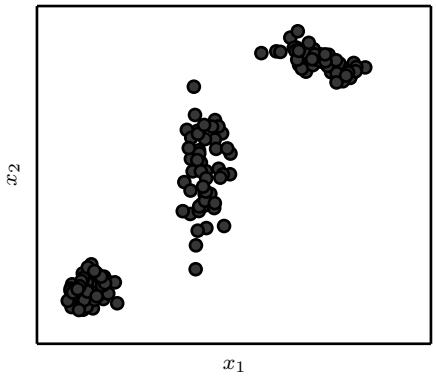
3.10. Użyteczne cechy elementarnych funkcji

Często podczas pracy z rozkładami prawdopodobieństwa, zwłaszcza rozkładami prawdopodobieństwa używanymi w modelach deep learning, pojawiają się pewne funkcje. Jedną z nich jest funkcja logistyczno-sigmoidalna:

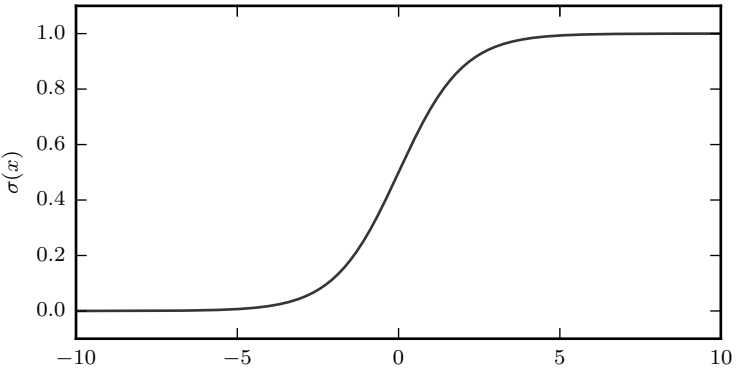
$$\sigma(x) = \frac{1}{1 + \exp(-x)}. \tag{3.30}$$

Sigmoida logistyczna jest powszechnie używana do pokazania parametru ϕ rozkładu Bernoulliego, gdyż jej zakresem jest $(0, 1)$, co odpowiada dopuszczalnemu zakresowi wartości parametru ϕ. Funkcję tę pokazano na rysunku 3.3.

Funkcja ulega nasyceniu, gdy jej argument jest mocno dodatni lub mocno ujemny, co oznacza, że funkcja staje się bardzo płaska i nieczuła na niewielkie zmiany argumentu wejściowego.



Rysunek 3.2. Przyk^{a}ady mieszanego modelu Gaussa. W tym przyk^{a}adzie mamy trzy sk^{a}adniki. Od lewej do prawej: pierwszy sk^{a}adnik ma izotropow¡ macierz kowariancji, co oznacza, »e ma t¦ sam¡ warto±¢ wariancji w ka»dym kierunku; drugi ma diagonaln¡ macierz kowariancji, co oznacza, »e mo»na sterowa¢ wariancjami oddzielnie wzd^{a}u» ka»dej osi w danym kierunku (w tym przyk^{a}adzie mamy wi¦ksz¡ wariancj¦ wzd^{a}u» osi x_2 ni» wzd^{a}u» osix_1); trzeci sk^{a}adnik ma pe^{a}n¡ macierz kowariancji, co umo»liwia oddzielne sterowanie wariancj¡ wzd^{a}u» arbitralnie wybranych podstawowych kierunków



Rysunek 3.3. Sigmoida logistyczna

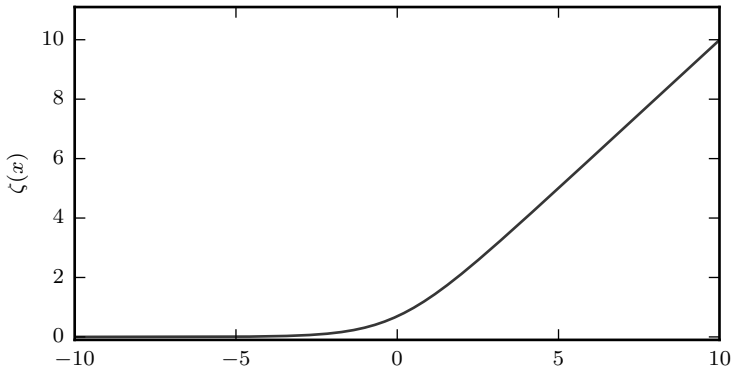
Inną często spotykaną funkcją jest **funkcja softplus** (Dugas et al. 2001):

$$\zeta(x) = \log\left(1 + \exp(x)\right). \tag{3.31}$$

Funkcja softplus może być wykorzystana do tworzenia parametru β lub σ rozkładu normalnego, gdyż jej zakresem jest przedział $(0, \infty)$. Pojawia się też często podczas przekształcania wyrażeń związanych z sigmoidami. Nazwa „softplus" wynika stąd, że jest wygładzoną, czyli „zmiękczoną" wersją funkcji:

$$x^{+} = \max(0, x). \tag{3.32}$$

Jej wykres widać na rysunku 3.4.



Rysunek 3.4. Funkcja softplus

Poniższe właściwości są także użyteczne, więc warto je zapamiętać:

$$\sigma(x) = \frac{\exp(x)}{\exp(x) + \exp(0)} \tag{3.33}$$

$$\frac{d}{dx}\sigma(x) = \sigma(x)(1 - \sigma(x)) \tag{3.34}$$

$$1 - \sigma(x) = \sigma(-x) \tag{3.35}$$

$$\log \sigma(x) = -\zeta(-x) \tag{3.36}$$

$$\frac{d}{dx}\zeta(x) = \sigma(x) \tag{3.37}$$

$$\forall x \in (0, 1),\ \sigma^{-1}(x) = \log\left(\frac{x}{1-x}\right) \tag{3.38}$$

$$\forall x > 0,\ \zeta^{-1}(x) = \log\left(\exp(x) - 1\right) \tag{3.39}$$

$$\zeta(x) = \int_{-\infty}^{x} \sigma(y)dy \tag{3.40}$$

$$\zeta(x) - \zeta(-x) = x \tag{3.41}$$

Funkcja $\sigma^{-1}(x)$ jest w statystyce nazywana **logit**, ale ten termin rzadko występuje w systemach uczących się.

W równaniu 3.41 podano dodatkowe uzasadnienie używania nazwy softplus. Jest to bowiem wygładzona wersja **dodatniej części funkcji** $x^+ = \max\{0, x\}$. Dodatnia część funkcji jest odpowiednikiem **ujemnej części funkcji** $x^- = \max\{0, x\}$. Aby otrzymać gładką funkcję, która jest analogiczna do części ujemnej, można użyć $\xi(-x)$. Podobnie jak x można otrzymać z jej dodatniej części i jej ujemnej części przez tożsamość $x^+ - x^- = \mathrm{x}$, można też otrzymać x za pomocą tego samego związku między $\xi(x)$ a $\xi(-x)$, co pokazano w równaniu 3.41.

3.11. Prawo Bayesa

Często znamy $P(\mathrm{y} \mid \mathrm{x})$, a chcemy poznać $P(\mathrm{x} \mid \mathrm{y})$. Na szczęście, jeśli znamy także $P(\mathrm{x})$, możemy wyznaczyć pożądaną ilość za pomocą **prawa Bayesa**:

$$P(\mathrm{x} \mid \mathrm{y}) = \frac{P(\mathrm{x})P(\mathrm{y} \mid \mathrm{x})}{P(\mathrm{y})}. \tag{3.42}$$

Zauważmy, że choć $P(\mathrm{y})$ pojawia się we wzorze, można zwykle obliczyć $P(\mathrm{y}) = \sum_x P(\mathrm{y} \mid x)P(x)$, więc nie trzeba wychodzić od znajomości $P(\mathrm{y})$.

Prawo Bayesa można bezpośrednio wyprowadzić z definicji prawdopodobieństwa warunkowego, ale warto znać nazwę tego wzoru, która pojawia się w wielu tekstach. Funkcję nazwano od nazwiska wielebnego Thomasa Bayesa, który pierwszy odkrył szczególny przypadek wzoru. Ogólna wersja, którą tu prezentujemy, została niezależnie opracowana przez Pierre-Simona Laplace'a.

3.12. Techniczne szczegóły zmiennych ciągłych

Poprawne formalne zrozumienie ciągłych zmiennych losowych i funkcji gęstości prawdopodobieństwa wymaga rozwinięcia teorii prawdopodobieństwa jako gałęzi matematyki znanej jako **teoria miary**. Teoria miary wykracza poza zakres tej książki, ale możemy krótko naszkicować zagadnienia, jakie teoria miary pomaga rozwiązywać.

W punkcie 3.3.2 pokazano, że prawdopodobieństwo ciągłych wartości wektorowych $\mathbf{x}$ leżących w pewnym zbiorze $\mathbb{S}$ jest dane całką funkcji $p(\boldsymbol{x})$ na zbiorze $\mathbb{S}$. Niektóre z wybranych zbiorów $\mathbb{S}$ mogą prowadzić do paradoksów. Na przykład można zbudować dwa zbiory $\mathbb{S}_1$ i $\mathbb{S}_2$ takie, że $p(\boldsymbol{x} \in \mathbb{S}_1) + p(\boldsymbol{x} \in \mathbb{S}_2) > 1$, ale $\mathbb{S}_1 \bigcap \mathbb{S}_2 = \phi$. Te zbiory są zbudowane z mocnym wykorzystaniem nieskończonej dokładności liczb rzeczywistych, na przykład przez tworzenie zbiorów o kształcie fraktalnym lub zbiorów, które są zdefiniowane przez transformację zbioru liczb wymiernych[2]. Jednym z kluczowych wartości teorii miary jest zapewnienie charakterystyki zbioru zbiorów, dla których możemy wyznaczyć prawdopodobieństwo bez napotykania na paradoksy. W tej książce całkujemy tylko zbiory o względnie prostym opisie, więc aspekty teorii miary nie są nigdy przedmiotem większej troski.

Dla naszych celów teoria miary jest bardziej użyteczna do opisu twierdzeń, które odnoszą się do większości punktów w $\mathbb{R}^n$, ale nie mają zastosowania do niektórych przypadków brzegowych. Teoria miary podaje ścisły sposób opisu, kiedy zbiór punktów jest pomijalnie mały. Taki zbiór określa się jako zbiór

[2]Paradoks Banacha-Tarskiego daje dobry przykład tego rodzaju zbiorów.

z **miarą zero**. W tej książce nie przedstawiamy tego pojęcia formalnie. Dla naszych celów wystarczy, aby zrozumieć intuicyjnie, że zbiór z miarą zero nie zajmuje objętości w przestrzeni, która jest przedmiotem naszej analizy. Na przykład w ramach $\mathbb{R}^2$ prosta ma miarę zero, a wypełniony wielokąt ma miarę dodatnią. Podobnie pojedynczy punkt ma miarę zero. Każda suma policzalnej liczby zbiorów, z których każdy ma miarę zero, także ma miarę zero (więc zbiór wszystkich liczb wymiernych ma przykładowo miarę zero).

Inne użyteczne terminy z teorii miary można znaleźć **niemal wszędzie**. Właściwość, która ma sens niemal wszędzie, utrzymuje się w całej przestrzeni poza zbiorem mającym miarę 0. Ponieważ wyjątki zajmują pomijalną ilość przestrzeni, mogą w wielu zastosowaniach zostać bezpiecznie zignorowane. Niektóre ważne wyniki w teorii prawdopodobieństwa są prawdziwe dla wszystkich wartości dyskretnych, ale są określone tylko dla wartości ciągłych.

Inny techniczny szczegół zmiennych ciągłych wiąże się z obsługą ciągłych zmiennych losowych, które są funkcjami wzajemnie deterministycznymi. Przypuśćmy, że mamy dwie zmienne losowe $\mathbf{x}$ i $\mathbf{y}$, takie, że $\boldsymbol{y} = g(\boldsymbol{x})$, gdzie g jest przekształceniem nieodwracalnym, ciągłym i różniczkowalnym. Można by się spodziewać, że $p_y(\boldsymbol{y}) = p_x(g^{-1}(\boldsymbol{y}))$. Ale okazuje się, że tak nie jest. Prosty przykład: przypuśćmy, że mamy skalarne zmienne losowe x i y. Przypuśćmy dalej, że $y = \frac{\mathrm{x}}{2}$ i $\mathrm{x} \sim U(0, 1)$. Jeśli skorzystamy z zasady, że $p_y(y) = p_x(2y)$, to p_y będzie miało wartość 0 wszędzie poza przedziałem $[0, \frac{1}{2}]$, a w tym przedziale będzie miało wartość 1. Oznacza to:

$$\int p_y(y)dy = \frac{1}{2}, \tag{3.43}$$

co narusza definicję rozkładu prawdopodobieństwa. To częsty błąd. Problem w tym podejściu stanowi fakt, że nie bierze pod uwagę zakłócenia przestrzeni wprowadzonego przez funkcję g. Przypomnijmy, że prawdopodobieństwo, że $\boldsymbol{x}$ leży w nieskończenie małym obszarze o objętości $\delta\boldsymbol{x}$, jest opisane przez $p(\boldsymbol{x})\delta\boldsymbol{x}$. Ponieważ g może rozszerzać lub zawężać przestrzeń, nieskończenie mała objętość otaczająca $\boldsymbol{x}$ w przestrzeni $\boldsymbol{x}$ może mieć inną objętość w przestrzeni $\boldsymbol{y}$.

Aby zobaczyć, jak naprawić ten problem, wróćmy do przypadku skalarnego. Musimy zachować właściwość:

$$|p_y(g(x))dy| = |p_x(x)dx|. \tag{3.44}$$

Rozwiązując ją, otrzymujemy:

$$p_y(y) = p_x(g^{-1}(y)) \left| \frac{\partial x}{\partial y} \right| \tag{3.45}$$

lub ekwiwalentnie:

$$p_x(x) = p_y(g(x)) \left| \frac{\partial g(x)}{\partial x} \right|. \tag{3.46}$$

Przy większej wymiarowości pochodna uogólnia wyznacznik **macierzy Jacobiego** – macierzy przy $J_{i,j} = \frac{\partial x_i}{\partial y_j}$. Stąd dla wektorów $\boldsymbol{x}$ i $\boldsymbol{y}$ o wartościach rzeczywistych:

$$p_x(\boldsymbol{x}) = p_y(g(\boldsymbol{x})) \left| \det \left(\frac{\partial g(\boldsymbol{x})}{\partial \boldsymbol{x}} \right) \right|. \tag{3.47}$$

3.13. Teoria informacji

Teoria informacji to gałąź matematyki stosowanej, która obraca się wokół określenia, ile informacji znajduje się w sygnale. Powstała w celu analizy wysyłania komunikatów w dyskretnych alfabetach przez kanał pełen szumów, jak transmisje radiowe. W tym kontekście teoria informacji mówi, jak zaprojektować optymalne kody i wyznaczyć oczekiwaną długość komunikatów próbkowanych według specjalnych rozkładów prawdopodobieństwa za pomocą różnych schematów kodowania. W kontekście systemów uczących się możemy zastosować teorię informacji do zmiennych ciągłych, w których nie mają zastosowania niektóre z tych interpretacji długości komunikatu. Ta dziedzina jest podstawą w wielu obszarach elektryki i nauk komputerowych. W książce korzystamy z kilku kluczowych pojęć z teorii informacji, aby scharakteryzować rozkłady prawdopodobieństwa lub ocenić podobieństwa między rozkładami prawdopodobieństwa. Więcej szczegółów na temat teorii informacji można znaleźć w Cover and Thomas (2006) lub MacKay (2003).

Intuicyjnie rozumiemy teorię informacji tak, że wiedza o tym, iż zaszło mało prawdopodobne zdarzenie, niesie więcej informacji niż to, że zaszło zdarzenie prawdopodobne. Komunikat mówiący „dziś wzeszło słońce" niesie tak mało informacji, że nie warto go wysyłać, natomiast komunikat „dziś rano było zaćmienie słońca" ma dużą zawartość informacyjną.

Warto określić ilościowo informacje tak, aby sformalizować to intuicyjne podejście.

- Zdarzenia prawdopodobne powinny mieć małą zawartość informacyjną, a w skrajnych przypadkach zdarzenia, które na pewno zajdą, nie powinny mieć jakiejkolwiek zawartości informacyjnej.
- Zdarzenia mniej prawdopodobne powinny mieć wyższą zawartość informacyjną.
- Zdarzenia niezależne powinny być informacyjnie addytywne. Na przykład stwierdzenie, że rzut monetą dał dwa razy orła, powinno zawierać dwa razy więcej informacji niż stwierdzenie, że orzeł wypadł tylko raz.

Aby spełnić te trzy właściwości/własności, definiujemy **własną informację** zdarzenia $\mathrm{x} = x$ jako:

$$I(x) = -\log P(x). \tag{3.48}$$

W tej książce log zawsze oznacza logarytm naturalny, czyli o podstawie e. Nasza definicja $\boldsymbol{I}(\mathrm{x})$ jest więc zapisana w jednostkach nat. Jeden nat to ilość informacji uzyskana przez zaobserwowanie zdarzenia o prawdopodobieństwie $\frac{1}{e}$. W innych tekstach używane są logarytmy o podstawie 2 oraz jednostki nazywane **bitami** (w literaturze anglosaskiej czasem występują jako **shannon**); informacja mierzona w bitach stanowi przeskalowanie informacji mierzonej w natach.

Gdy x jest ciągłe, to przez analogię używamy tej samej definicji informacji, ale niektóre właściwości w przypadku dyskretnym zostają zagubione. Na przykład zdarzenie o jednostkowej gęstości ma nadal zerową informację, choć nie jest zdarzeniem, które zdarzy się na pewno.

Informacje własne dotyczą tylko pojedynczego wyniku. Możemy ocenić ilościowo poziom niepewności całego rozkładu prawdopodobieństwa za pomocą **entropii Shannona:**

$$H(\mathrm{x}) = \mathbb{E}_{\mathrm{x}\sim P}[I(x)] = -\mathbb{E}_{\mathrm{x}\sim P}[\log P(x)], \tag{3.49}$$

oznaczanej także jako $H(P)$. Innymi słowy, entropia Shannona rozkładu to oczekiwana ilość informacji w zdarzeniu losowanym z tego rozkładu. Daje dolną granicę liczby bitów (jeśli logarytm ma podstawę 2, w innym przypadku jednostki są inne) potrzebną średnio do zakodowania symboli wylosowanych z rozkładu P. Rozkłady, które są niemal deterministyczne (gdzie wynik jest niemal pewny), mają niską entropię; rozkłady bliższe rozkładu jednostajnego mają wysoką entropię. Pokazuje to rysunek 3.5. Gdy x jest ciągłe, entropia Shannona określana jest jako **entropia różniczkowa**.

Jeśli mamy dwa oddzielne rozkłady prawdopodobieństwa $P(\mathrm{x})$ i $Q(\mathrm{x})$ względem tej samej zmiennej losowej x, możemy zmierzyć, na ile się one różnią, za pomocą **dywergencji Kullbacka-Leiblera (KL)**:

$$D_{\mathrm{KL}}(P\|Q) = \mathbb{E}_{\mathrm{x}\sim P}\left[\log \frac{P(x)}{Q(x)}\right] = \mathbb{E}_{\mathrm{x}\sim P}\left[\log P(x) - \log Q(x)\right]. \tag{3.50}$$

W przypadku zmiennych dyskretnych mamy dodatkową ilość informacji (mierzoną w bitach, jeśli stosujemy logarytm przy podstawie 2, choć w systemach uczących się stosuje się zwykle naty i logarytmy naturalne), potrzebną do wysłania komunikatu zawierającego symbole wyprowadzone z rozkładu prawdopodobieństwa P, gdy korzystamy z kodu opracowanego na potrzeby minimalizacji długości komunikatu na podstawie rozkładu prawdopodobieństwa Q.