MATEMATYKA W DEEP LEARNINGU

CO MUSISZ WIEDZIEĆ, ABY ZROZUMIEĆ SIECI NEURONOWE

RONALD T. KNEUSEL

Tytuł oryginału: Math for Deep Learning: What You Need to Know
to Understand Neural Networks

Tłumaczenie: Filip Kamiński

ISBN: 978-83-289-1016-4

Copyright © 2022 by Ronald T. Kneusel. Title of English-language original:
Math for Deep Learning: What You Need to Know to Understand Neural Networks,
ISBN 9781718501904, published by No Starch Press Inc. 245 8th Street,
San Francisco, California United States 94103.

The Polish-language 1st edition Copyright © 2024 by Helion S.A. under license
by No Starch Press Inc. All rights reserved.

All rights reserved. No part of this book may be reproduced or transmitted in any
form or by any means, electronic or mechanical, including photocopying, recording
or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości
lub fragmentu niniejszej publikacji w jakiejkolwiek postaci jest zabronione.
Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie
książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie
praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi
bądź towarowymi ich właścicieli.

Autor oraz wydawca dołożyli wszelkich starań, by zawarte w tej książce informacje
były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich
wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych
lub autorskich. Autor oraz wydawca nie ponoszą również żadnej odpowiedzialności
za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

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

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
https://ftp.helion.pl/przyklady/matdee.zip

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

Printed in Poland.

- Kup książkę
- Poleć książkę
- Oceń książkę

- Księgarnia internetowa
- Lubię to! » Nasza społeczność

Spis treści

PRZEDMOWA **13**

WPROWADZENIE **19**

1
PRZYGOTOWANIE ŚRODOWISKA PRACY **23**

Instalowanie zestawu narzędzi 24
Linuks 24
macOS 25
Windows 26
NumPy 27
Definiowanie tablic 27
Typy danych 28
Tablice dwuwymiarowe 29
np.zeros i np.ones 29
Zaawansowane indeksowanie 30
Odczyt i zapis na dysku 32
SciPy 33
Matplotlib 34
Scikit-learn 36
Podsumowanie 38

2
PRAWDOPODOBIEŃSTWO **39**

Podstawowe koncepcje 40
Przestrzeń próbek i zdarzenia 40
Zmienne losowe 41
Ludzie nie radzą sobie z prawdopodobieństwem 41
Reguły prawdopodobieństwa 43
Prawdopodobieństwo zdarzenia 44
Reguła dodawania 47
Reguła mnożenia 47
Ponowne spojrzenie na regułę dodawania 48

Kup ksi k   Pole ksi k

Paradoks dnia urodzin 49
Prawdopodobieństwo warunkowe 52
Prawdopodobieństwo całkowite 54
Prawdopodobieństwo łączne i brzegowe 55
Tabele prawdopodobieństwa łącznego 56
Reguła łańcuchowa dla prawdopodobieństwa 60
Podsumowanie 62

3
WIĘCEJ PRAWDOPODOBIEŃSTWA 63
Rozkłady prawdopodobieństwa 63
Histogramy i prawdopodobieństwa 64
Dyskretne rozkłady prawdopodobieństwa 68
Ciągłe rozkłady prawdopodobieństwa 73
Centralne twierdzenie graniczne 77
Prawo wielkich liczb 80
Twierdzenie Bayesa 80
Rak czy nie rak 82
Aktualizacja prawdopodobieństwa a priori 83
Twierdzenie Bayesa w uczeniu maszynowym 84
Podsumowanie 87

4
STATYSTYKA 89
Rodzaje danych 90
Dane nominalne 90
Dane porządkowe 90
Dane interwałowe 90
Dane ilorazowe 91
Wykorzystanie danych nominalnych w uczeniu głębokim 92
Statystyki podsumowujące 92
Średnie i mediana 93
Miary zmienności 97
Kwantyle i wykresy pudełkowe 100
Braki w danych 106
Korelacja 109
Współczynnik korelacji Pearsona 109
Korelacja Spearmana 113
Testowanie hipotez 115
Hipotezy 117
Test t 118
Test U Manna-Whitneya 123
Podsumowanie 125

Kup książkę Poleć książkę

5
ALGEBRA LINIOWA ... 127
Skalary, wektory, macierze i tensory ... 128
Skalary ... 128
Wektory ... 128
Macierze ... 129
Tensory ... 130
Arytmetyka tensorów ... 133
Operacje tablicowe ... 133
Operacje wektorowe ... 136
Mnożenie macierzy ... 144
Iloczyn Kroneckera ... 149
Podsumowanie ... 151

6
WIĘCEJ ALGEBRY LINIOWEJ ... 153
Macierze kwadratowe ... 154
Dlaczego macierze kwadratowe? ... 154
Transpozycja, ślad i potęgowanie ... 155
Specjalne macierze kwadratowe ... 157
Macierz jednostkowa ... 158
Wyznaczniki ... 160
Odwrotności ... 163
Macierze symetryczne, ortogonalne i unitarne ... 165
Określoność macierzy symetrycznych ... 166
Wektory i wartości własne ... 166
Znajdowanie wartości i wektorów własnych ... 167
Normy wektorowe i miary odległości ... 170
L-normy oraz miary odległości ... 170
Macierze kowariancji ... 171
Odległość Mahalanobisa ... 174
Dywergencja Kullbacka-Leiblera ... 176
Analiza głównych składowych ... 179
Rozkład według wartości osobliwych i pseudoodwrotności ... 183
SVD w akcji ... 184
Dwa zastosowania ... 185
Podsumowanie ... 188

Kup książkę Poleć książkę

7
RACHUNEK RÓŻNICZKOWY ... 189
Nachylenie ... 190
Pochodne ... 191
Definicja formalna ... 191
Podstawowe zasady ... 193
Funkcje trygonometryczne ... 197
Funkcje wykładnicze i logarytmy ... 199
Minima i maksima funkcji ... 202
Pochodne cząstkowe ... 206
Mieszane pochodne cząstkowe ... 207
Reguła łańcuchowa dla pochodnych cząstkowych ... 208
Gradienty ... 210
Obliczanie gradientu ... 210
Wizualizacja gradientu ... 212
Podsumowanie ... 215

8
MACIERZOWY RACHUNEK RÓŻNICZKOWY ... 217
Formuły ... 218
Funkcja wektorowa z argumentem skalarnym ... 219
Funkcja skalarna z argumentem wektorowym ... 220
Funkcja wektorowa przyjmująca wektor ... 221
Funkcja macierzowa przyjmująca skalar ... 221
Funkcja skalarna przyjmująca macierz ... 222
Tożsamości ... 223
Funkcja skalarna z argumentem wektorowym ... 223
Funkcja wektorowa z argumentem skalarnym ... 225
Funkcja wektorowa przyjmująca wektor ... 226
Funkcja skalarna przyjmująca macierz ... 227
Macierze Jacobiego i hesjany ... 228
Macierze Jacobiego ... 228
Hesjany ... 235
Wybrane przykłady z macierzowego rachunku różniczkowego ... 241
Pochodna operacji na elementach ... 241
Pochodna funkcji aktywacji ... 242
Podsumowanie ... 244

Kup ksi k Pole ksi k

9
PRZEPŁYW DANYCH W SIECIACH NEURONOWYCH 245
Reprezentacja danych 246
Tradycyjne sieci neuronowe 246
Głębokie sieci konwolucyjne 247
Przepływ danych w tradycyjnych sieciach neuronowych 249
Przepływ danych w konwolucyjnych sieciach neuronowych 253
Konwolucja 254
Warstwy konwolucyjne 259
Warstwy łączące 262
Warstwy w pełni połączone 263
Przepływ danych w konwolucyjnej sieci neuronowej 264
Podsumowanie 267

10
PROPAGACJA WSTECZNA 269
Czym jest propagacja wsteczna? 270
Ręczne przeprowadzanie propagacji wstecznej 271
Pochodne cząstkowe 272
Zamiana formuł na kod w Pythonie 274
Uczenie i testowanie modelu 278
Propagacja wsteczna w sieciach w pełni połączonych 280
Wsteczna propagacja błędu 281
Obliczanie pochodnych cząstkowych wag i wyrazów wolnych 283
Implementacja w Pythonie 285
Korzystanie z implementacji 290
Grafy obliczeniowe 292
Podsumowanie 295

11
METODA GRADIENTU PROSTEGO 297
Podstawowa idea 298
Jednowymiarowa metoda gradientu prostego 298
Metoda gradientu prostego w dwóch wymiarach 302
Stochastyczna metoda gradientu prostego 308
Pęd 311
Czym jest pęd? 311
Pęd w jednym wymiarze 312
Pęd w dwóch wymiarach 314
Uczenie modeli za pomocą metod z pędem 315
Pęd Niestierowa 321

Kup ksi k Pole ksi k

Adaptacyjna metoda gradientu prostego ... 324
RMSprop ... 324
Adagrad i Adadelta ... 325
Adam ... 326
Kilka uwag na temat metod optymalizacji ... 327
Podsumowanie ... 329
Epilog ... 329

DODATEK. CO DALEJ? ... 331

SKOROWIDZ ... 335

Kup książkę Poleć książkę

5

Algebra liniowa

Z formalnego punktu widzenia algebra liniowa to nauka o równaniach liniowych, czyli takich, w których każda niewiadoma występuje co najwyżej w pierwszej potędze. Natomiast w naszych zastosowaniach pod pojęciem **algebra liniowa** należy rozumieć naukę o wielowymiarowych obiektach matematycznych, takich jak wektory i macierze, oraz wykonywanych na nich operacjach. To właśnie do tych celów wykorzystuje się zazwyczaj algebrę liniową w uczeniu głębokim. Reguły te są wykorzystywane również w programach implementujących algorytmy głębokiego uczenia. Decydując się na ograniczenie omówienia algebry jedynie do tych kwestii, odrzucam ogromną ilość fascynującej matematyki, ale mam nadzieje, że mi to wybaczysz, biorąc pod uwagę to, że celem tej książki jest przedstawienie elementów matematyki wykorzystywanych w uczeniu głębokim.

W tym rozdziale przedstawię obiekty matematyczne wykorzystywane w uczeniu głębokim, czyli w szczególności skalary, wektory, macierze i tensory. Jak się wkrótce przekonasz, wszystkie te obiekty są w rzeczywistości tensorami różnych rzędów. Najpierw przedstawię je z perspektywy notacji matematycznej, a następnie przeprowadzimy kilka

Kup ksi k Pole ksi k

eksperymentów za pomocą NumPy. NumPy to pakiet wzbogacający Pythona o wielowymiarowe tablice. Tablice te to dobre, choć niepełne analogi obiektów matematycznych, z którymi będziemy pracowali w tym rozdziale.

Większą część rozdziału poświęcę arytmetyce tensorów, która ma fundamentalne znaczenie w uczeniu głębokim. Większość wysiłków związanych z implementacją wysoce wydajnych rozwiązań uczenia głębokiego sprowadza się do znalezienia sposobów na wydajne wykonywanie operacji na tensorach.

Skalary, wektory, macierze i tensory

Czas przedstawić naszych bohaterów. Powiążę ich ze zmiennymi Pythona i tablicami NumPy, aby pokazać, jak wykorzystuje się je w kodzie. Następnie pokażę Ci przydatną geometryczną interpretację tensorów.

Skalary

Nawet jeśli nie słyszałeś wcześniej tego słowa, wiesz, czym jest skalar, od dnia, w którym nauczyłeś się liczyć. **Skalar** to po prostu liczba, np. 7, 42 lub π. W formułach matematycznych skalar będę oznaczał tak jak zwykłe zmienne, czyli za pomocą liter, np. x. Z punktu widzenia komputera skalar to po prostu zmienna numeryczna:

```
>>> s = 66
>>> s
66
```

Wektory

Wektor to jednowymiarowa tablica liczb. W matematyce wektor ma orientację, która może być pozioma lub pionowa. Wektor o orientacji poziomej nazywamy **wektorem wierszowym**. Na przykład

$$\boldsymbol{x} = [x_0 \; x_1 \; x_2] \tag{5.1}$$

jest wektorem wierszowym zawierającym trzy elementy lub trzy komponenty. Zauważ, że wektory oznaczamy za pomocą małych pogrubionych liter, w tym przypadku $\boldsymbol{x}$.

W matematyce najczęściej zakłada się, że wektory są **wektorami kolumnowymi**:

$$\boldsymbol{y} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} \tag{5.2}$$

Zdefiniowany powyżej wektor $\boldsymbol{y}$ zawiera cztery elementy, które czynią z niego wektor czterowymiarowy (4D). Zauważ, że w równaniu 5.1 zastosowałem nawiasy kwadratowe, a w równaniu 5.2 okrągłe. Dopuszczalne są obie postacie.

Kup ksi k Pole ksi k

W kodzie wektory zapisuje się najczęściej w postaci jednowymiarowych tablic:

```
>>> import numpy as np
>>> x = np.array([1,2,3])
>>> print(x)
[1 2 3]
>>> print(x.reshape((3,1)))
[[1]
 [2]
 [3]]
```

W powyższym kodzie za pomocą funkcji `reshape` przekształciłem trzyelementowy wektor wierszowy w wektor kolumnowy składający się z trzech wierszy i jednej kolumny.

Elementy wektora często traktuje się jako odległości od środka jakiegoś układu współrzędnych. Na przykład wektor trójwymiarowy możemy wykorzystać do przedstawienia punktu leżącego w przestrzeni trójwymiarowej. W wektorze

$$\boldsymbol{x} = [x, y, z]$$

x może być położeniem na osi x, y położeniem na osi y, a z położeniem na osi z. Są to tak zwane współrzędne kartezjańskie. Współrzędne te pozwalają w unikalny sposób opisać wszystkie punkty leżące w przestrzeni trójwymiarowej.

Natomiast w uczeniu głębokim, a właściwie to w całym uczeniu maszynowym, elementy wektorów nie są ze sobą zazwyczaj powiązane w żaden ściśle geometryczny sposób. W tych zastosowaniach elementy wektorów wykorzystuje się raczej do reprezentowania **cech** (ang. *features*) jakiejś próbki, która ma być wykorzystana przez model do wyprodukowania wyniku, takiego jak etykieta klasy lub wartość regresji. Wektor reprezentujący zbiór cech nazywamy **wektorem cech** i czasami rozpatrujemy jego interpretację geometryczną. Niektóre modele uczenia maszynowego, takie jak algorytm k-najbliższych sąsiadów, traktują wektor jako współrzędne punktu w przestrzeni geometrycznej.

Osoby zajmujące się uczeniem głębokim często wspominają o **przestrzeni cech** problemu. Przestrzeń cech to zbiór wszystkich możliwych danych wejściowych. Zbiór uczący musi dobrze odzwierciedlać przestrzeń cech danych wejściowych, z którymi model spotka się podczas jego stosowania w praktyce. W tym sensie wektor cech jest punktem w n-wymiarowej przestrzeni (n jest liczbą cech w wektorze cech).

Macierze

Macierz to dwuwymiarowa tablica liczb:

$$\boldsymbol{A} = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \end{bmatrix}$$

Elementy $\boldsymbol{A}$ są indeksowane numerem wiersza i kolumny. Macierz $\boldsymbol{A}$ ma trzy wiersze i cztery kolumny. O takiej macierzy mówimy, że jest to macierz o wymiarach 3×4. Zauważ, że indeksy elementów z macierzy $\boldsymbol{A}$ zaczynają się od 0. W literaturze matematycznej często indeksuje się od 1, ale coraz częściej stosuje się także indeksowanie od 0, które

pozwala uniknąć różnic pomiędzy zapisem matematycznym a reprezentacją macierzy na komputerze. Macierze oznaczamy za pomocą dużych pogrubionych liter, np. $\boldsymbol{A}$.

Do reprezentowania macierzy w kodzie wykorzystuje się tablice dwuwymiarowe:

```
>>> A = np.array([[1,2,3],[4,5,6],[7,8,9]])
>>> print(A)
[[1 2 3]
 [4 5 6]
 [7 8 9]]
>>> print(np.arange(12).reshape((3,4)))
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```

W Pythonie dostęp do elementu a_{12} macierzy $\boldsymbol{A}$ jest możliwy za pomocą instrukcji `A[1,2]`. Zauważ, że po wyświetleniu tablic na ekranie widoczna jest dodatkowa para nawiasów ([i]). Za pomocą tych nawiasów NumPy mówi nam, że możemy traktować macierz jako wektor wierszowy, którego elementami są wektory. W języku Python oznacza to, że macierz można traktować jako listę list o tej samej długości. Oczywiście to właśnie w ten sposób zdefiniowaliśmy macierz `A` na samym początku powyższego listingu.

Wektory możemy traktować jako macierze z jednym wierszem lub jedną kolumną. Trzyelementowy wektor kolumnowy jest macierzą o wymiarach 3×1 — ma trzy wiersze i jedną kolumnę. Czteroelementowy wektor wierszowy zachowuje się jak macierz 1×4 — ma jeden wiersz i cztery kolumny. Obserwację tę wykorzystamy w dalszej części rozdziału.

Tensory

Skalar nie ma wymiarów, wektor ma jeden wymiar, a macierz dwa wymiary. Jak możesz podejrzewać, nie jest to koniec. Obiekt matematyczny o więcej niż dwóch wymiarach nazywa się potocznie **tensorem**. Jeżeli pojawi się taka konieczność, to tensory w tej książce będę oznaczał dużą literą T zapisaną czcionką bezszeryfową.

Liczbę wymiarów w tensorze nazywamy jego **rzędem** (nie mylić z rzędem macierzy). Tensor 3D ma rząd równy trzy. Macierz to tensor drugiego rzędu, wektor to tensor pierwszego rzędu, a skalar to tensor rzędu zero. W rozdziale 9. poświęconym przepływowi danych przez głębokie sieci neuronowe przekonasz się, że wiele narzędzi do uczenia głębokiego wykorzystuje tensory czwartego i wyższych rzędów.

W Pythonie tensory można reprezentować za pomocą tablic NumPy o trzech i większej liczbie wymiarów. Na przykład na poniższym listingu definiuję w Pythonie tensor 3. rzędu:

```
>>> t = np.arange(36).reshape((3,3,4))
>>> print(t)
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

 [[12 13 14 15]
```

```
 [16 17 18 19]
 [20 21 22 23]]

[[24 25 26 27]
 [28 29 30 31]
 [32 33 34 35]]]
```

Za pomocą `np.arange` tworzę wektor `t` zawierający 36 elementów będących liczbami od 0 do 35. Następnie za pomocą metody `reshape` w tej samej linii przekształcam go w tensor o wymiarach 3×3×4 (liczący $3 \cdot 3 \cdot 4 = 36$ elementów). Jednym ze sposobów myślenia o tensorze 3×3×4 jest potraktowanie go jako stosu, na którym znajdują się trzy obrazki o wymiarach 3×4. Jeśli będziesz pamiętał o tej interpretacji, to następujące stwierdzenia powinny mieć sens:

```
>>> print(t[0])
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
>>> print(t[0,1])
[4 5 6 7]
>>> print(t[0,1,2])
6
```

Żądanie `t[0]` zwróci ze stosu pierwszy *obrazek* o wymiarach 3×4. Instrukcja `t[0,1]` powinna zatem zwrócić drugi wiersz z pierwszego obrazka. Tak też jest. W ostatniej instrukcji chcę dostać się do pojedynczego elementu `t`. W tym celu podaję numer obrazka (`0`), numer wiersza (`1`) i interesujący mnie element z tego wiersza (`2`).

Przypisywanie kolejnych wymiarów tensora do coraz mniejszych podzbiorów jest wygodnym sposobem na zapamiętanie znaczenia wymiarów. Na przykład w następujący sposób możemy zdefiniować tensor 5. rzędu:

```
>>> w = np.zeros((9,9,9,9,9))
>>> w[4,1,2,0,1]
0.0
```

Co w takim przypadku oznacza `w[4,1,2,0,1]`? Dokładna interpretacja zależy od zastosowania. Jeżeli potraktujemy `w` jako regał z książkami, to pierwszy indeks oznacza wybór półki, a drugi wybór książki znajdującej się na wybranej półce. Następnie za pomocą trzeciego indeksu wybieram stronę w książce, a za pomocą czwartego linijkę na tej stronie. Za pomocą ostatniego indeksu wybieram słowo z wybranej linii. A zatem `w[4,1,2,0,1]` to prośba o drugie słowo z pierwszego wiersza trzeciej strony drugiej książki leżącej na piątej półce naszego regału (kolejność odczytu informacji to w tym przypadku od prawej do lewej).

Analogia z regałem ma swoje ograniczenia. Tablice NumPy mają niezmienne wymiary, co oznacza, że jeśli `w` jest regałem, to ma on dziewięć półek, a na każdej z nich znajduje się *dokładnie* dziewięć książek, z których każda ma dokładnie dziewięć stron liczących

Kup ksi k Pole ksi k

po dziewięć wierszy. Każdy z tych wierszy zawiera zaś dziewięć słów. W pamięci komputera zawartość tablic NumPy jest zapisywana w ciągłym obszarze pamięci, co wymusza ustalenie rozmiarów poszczególnych wymiarów już na etapie tworzenia tablicy. Wykonanie tej czynności i wybranie typu przechowywanych w tablicy danych (np. liczba całkowita bez znaku) pozwala na dostęp do poszczególnych elementów tablicy za pomocą prostej formuły pozwalającej wyznaczyć przesunięcie bitowe od adresu początku tablicy. To właśnie dzięki temu tablice NumPy działają o wiele szybciej od list Pythona.

Dowolny tensor o rzędzie mniejszym niż n możemy przedstawić w postaci tensora rzędu n poprzez ustawienie długości każdego z brakujących wymiarów na 1. Pokazałem Ci już przykład takiego działania, gdy wspomniałem, że m-elementowy wektor możemy potraktować jako macierz o wymiarach $1 \times m$ lub $m \times 1$. Tensor 1. rzędu (wektor) możemy przekształcić w tensor 2. rzędu (macierz) poprzez dodanie brakującego wymiaru o długości jeden.

Skrajnym przykładem takiego działania jest potraktowanie skalara (tensor rzędu 0) jako tensor 5. rzędu:

```
>>> t = np.array(42).reshape((1,1,1,1,1))
>>> print(t)
[[[[[42]]]]]
>>> t.shape
(1, 1, 1, 1, 1)
>>> t[0,0,0,0,0]
42
```

Na powyższym listingu przekształcam skalar 42 w tensor 5. rzędu (tablicę pięciowymiarową). Długość każdego wymiaru tego tensora to jeden. Zauważ, że NumPy mówi nam, że tensor `t` ma pięć wymiarów — liczbę 42 w danych wyjściowych otacza `[[[[[` oraz `]]]]]`. Zapytanie o kształt `t` potwierdza, że jest to pięciowymiarowy tensor. Ponieważ jest to tensor, dostęp do przechowywanej w nim wartości możemy uzyskać za pomocą `t[0,0,0,0,0]`. W dalszej części książki dość często korzystam ze sztuczki polegającej na dodawaniu nowych wymiarów o długości jeden. W NumPy istnieje też bezpośredni sposób na dodanie nowych wymiarów, który często stosuje się podczas pracy z narzędziami do uczenia głębokiego:

```
>>> t = np.array([[1,2,3],[4,5,6]])
>>> print(t)
[[1 2 3]
 [4 5 6]]
>>> w = t[np.newaxis,:,:]
>>> w.shape
(1, 2, 3)
>>> print(w)
[[[1 2 3]
  [4 5 6]]]
```


Kup ksi k Pole ksi k

Na powyższym listingu zamieniłem tensor 2. rzędu `t` (macierz) w tensor rzędu 3. Nowy wymiar o długości jeden utworzyłem za pomocą `np.newaxis`. Po dodaniu wymiaru `w.shape` zwraca `(1,2,3)`, a nie `(2,3)`, tak jak w przypadku `t`.

Tensory o rzędach do trzeciego włącznie mają swoje odpowiedniki geometryczne, które mogą okazać się pomocne w wizualizacji relacji pomiędzy nimi:

Rząd (liczba wymiarów)	Nazwa tensora	Odpowiednik geometryczny
0	skalar	punkt
1	wektor	prosta
2	macierz	płaszczyzna
3	tensor	bryła

W powyższej tabeli użyłem słowa *tensor* w potocznym znaczeniu. Tensor 3. rzędu nie ma żadnej powszechnie akceptowanej nazwy.

W tym podrozdziale zdefiniowaliśmy obiekty matematyczne występujące w uczeniu głębokim w postaci wielowymiarowych tablic, czyli formy, w jakiej implementuje się je w kodzie. Takie działanie wiązało się z odrzuceniem dużej ilości matematyki. Przedstawiłem Ci jedynie informacje niezbędne do zrozumienia uczenia głębokiego. W kolejnym podrozdziale pokażę Ci, jak korzysta się z tensorów w wyrażeniach matematycznych.

Arytmetyka tensorów

W tym rozdziale przedstawię szczegółowo operacje na tensorach ze specjalnym uwzględnieniem tensorów 1. rzędu (wektory) i 2. rzędu (macierze). Zakładam, że w tym momencie książki wiesz już, jak wykonuje się operacje na skalarach.

Zacznę od czegoś, co nazywam **operacjami tablicowymi**. Pod tym pojęciem rozumiem operacje na elementach, które narzędzia takie jak NumPy wykonują na tablicach o dowolnej liczbie wymiarów. Następnie przejdę do omówienia operacji wektorowych i przygotuję grunt pod prezentację kluczowego zagadnienia, jakim jest mnożenia macierzy. Na koniec omówię macierze blokowe.

Operacje tablicowe

Nasze dotychczasowe doświadczenia z NumPy pokazują, że wszystkie normalne operacje arytmetyczne wykonywane na skalarach przekładają się bezpośrednio na świat tablic wielowymiarowych. Do tych operacji zaliczamy m.in. dodawanie, odejmowanie, mnożenie, dzielenie i potęgowanie, a także wywoływanie funkcji na tablicy. We wszystkich tych przypadkach wywoływana operacja jest stosowana na każdym elemencie tablicy. W pozostałej części tego podrozdziału wykorzystam poniższe przykłady. Przykłady te pozwolą nam również przyjrzeć się regułom broadcastingu, o których jeszcze nie wspominałem.

Kup ksi k Pole ksi k

Zaczynam od zdefiniowania kilku tablic:

```
>>> a = np.array([[1,2,3],[4,5,6]])
>>> b = np.array([[7,8,9],[10,11,12]])
>>> c = np.array([10,100,1000])
>>> d = np.array([10,11])
>>> print(a)
[[1 2 3]
 [4 5 6]]
>>> print(b)
[[ 7  8  9]
 [10 11 12]]
>>> print(c)
[  10  100 1000]
>>> print(d)
[10 11]
```

W przypadku tablic o zgodnych wymiarach wykonywanie operacji na ich elementach jest dość proste:

```
>>> print(a+b)
[[ 8 10 12]
 [14 16 18]]
>>> print(a-b)
[[-6 -6 -6]
 [-6 -6 -6]]
>>> print(a*b)
[[ 7 16 27]
 [40 55 72]]
>>> print(a/b)
[[0.14285714 0.25       0.33333333]
[0.4        0.45454545 0.5       ]]
>>> print(b**a)
[[      7      64     729]
 [  10000  161051 2985984]]
```

Interpretacja wszystkich powyższych wyników nie stwarza problemów. W każdym przypadku NumPy zastosował wywołaną operację na wszystkich odpowiadających sobie elementach z każdej z tablic. Mnożenie odpowiadających sobie elementów z dwóch macierzy (`a` i `b`) to tak zwany **iloczyn Hadamarda**. (Nazwę tę spotyka się czasami w pracach poświęconych uczeniu głębokiemu).

Pakiet NumPy rozszerza ideę operacji wykonywanych na elementach o tak zwany **broadcasting**. Polega on na stosowaniu pewnych reguł (których przykłady zaraz Ci pokażę) pozwalających uzyskać sensowne wyniki w przypadku niezgodności wymiarów.

Z broadcastingiem spotkaliśmy się już w przypadku działań wykonywanych na tablicy i skalarze, w trakcie których to wartość skalarna była przekazywana do każdego elementu tablicy.

Kup ksi k Pole ksi k

W pierwszym przykładzie, pomimo tego, że a jest macierzą o wymiarach 2×3, dzięki broadcastingowi możemy wykonać operacje dwuargumentowe na tej tablicy i tablicy c, czyli trzyelementowym wektorze:

```
>>> print(a+c)
[[  11 102 1003]
 [  14 105 1006]]
>>> print(c*a)
[[  10 200 3000]
 [  40 500 6000]]
>>> print(a/c)
[[0.1   0.02  0.003]
 [0.4   0.05  0.006]]
```

W powyższym kodzie trzyelementowy wektor c został dopasowany do macierzy a o wymiarach 2×3. NumPy rozpoznał, że zarówno w przypadku a, jak i c drugi wymiar ma długość trzy. Dzięki temu możliwe jest zduplikowanie wartości w wektorze c i zamiana go w macierz, której wymiary będą pasowały do wymiarów a, co pozwoli nam na obliczenie wyniku. Tego typu operacje dość często pojawiają się w kodach algorytmów uczenia maszynowego, z których większość napisana jest w Pythonie. Czasami, aby zrozumieć, co się dzieje, warto poeksperymentować z interpreterem Pythona i zastanowić się nad wynikami.

Czy możemy zastosować broadcasting i dopasować dwuelementowy wektor d do macierzy a o wymiarach 2×3? Jeśli spróbujemy to zrobić w ten sam sposób co w przypadku a i c, to otrzymamy błąd:

```
>>> print(a+d)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: operands could not be broadcast together with shapes (2,3) (2,)
```

Reguły broadcastingu NumPy wymagają istnienia wymiarów o długości jeden. Kształt d to (2,) — jest to wektor dwuelementowy bez kierunku. Jeśli zmienimy kształt d tak, aby była to dwuwymiarowa tablica o wymiarach 2×1, NumPy poradzi sobie z tym działaniem:

```
>>> d = d.reshape((2,1))
>>> d.shape
(2, 1)
>>> print(a+d)
[[11 12 13]
 [15 16 17]]
```

Jak widzisz, NumPy dodał d do poszczególnych kolumn a.

Wróćmy do świata matematyki i przyjrzyjmy się działaniom na wektorach.

Kup książkę Poleć książkę

Operacje wektorowe

W kodzie wektory są reprezentowane w postaci ciągu liczb, które można interpretować jako wartości wzdłuż jakiegoś zbioru osi. W tym punkcie przedstawię kilka unikalnych operacji na wektorach.

Długość wektora

Z geometrycznego punktu widzenia wektory mają kierunek oraz długość. Wektory często rysuje się w postaci strzałek. Przykład rysunku z takimi strzałkami zobaczysz w rozdziale 6. Pierwszą operacją wektorową, której się przyjrzymy, będzie wyznaczanie długości wektora. Długość n-elementowego wektora $\boldsymbol{x}$ to

$$\| \boldsymbol{x} \| = \sqrt{x_0^2 + x_1^2 + \cdots + x_{n-1}^2} \tag{5.3}$$

W równaniu 5.3 z obu stron $\boldsymbol{x}$ widoczne są podwójne pionowe kreski oznaczające długość wektora. Często zamiast podwójnych kresek stosuje się pojedyncze. Jest to trochę mylące, ponieważ pojedyncze kreski oznaczają też wartość bezwzględną. Najczęściej to z kontekstu wynika, co mamy na myśli.

Skąd wzięło się równanie 5.3? Rozważmy wektor $\boldsymbol{x} = (x, y)$ z przestrzeni dwuwymiarowej. Jeśli x i y są długościami odpowiednio wzdłuż osi x i y, to długości te tworzą boki trójkąta prostokątnego. Długość przeciwprostokątnej tego trójkąta prostokątnego jest długością wektora. Jak ustalił Pitagoras, a nad długo przed nim Babilończycy, długość tej przeciwprostokątnej to $\sqrt{x^2 + y^2}$. Uogólnienie tej formuły na n-wymiarów daje równanie 5.3.

Wektory jednostkowe

Teraz gdy wiesz już, jak obliczyć długość wektora, mogę wprowadzić pewną użyteczną postać wektora nazywaną **wektorem jednostkowym**. Jeśli podzielimy elementy wektora przez jego długość, otrzymamy wektor skierowany w tym samym kierunku co pierwotny wektor, ale o długości jeden. To właśnie wektor jednostkowy. Dla wektora $\boldsymbol{v}$ wektor jednostkowy skierowany w tym samym kierunku to:

$$\hat{\boldsymbol{v}} = \frac{\boldsymbol{v}}{\| \boldsymbol{v} \|}$$

Daszek nad nazwą wektora to symbol wektora jednostkowego. Spójrz na konkretny przykład. Nasz przykładowy wektor to $\boldsymbol{v} = (2, -4, 3)$. Wektor jednostkowy skierowany w tym samym kierunku co $\boldsymbol{v}$ to

$$\hat{\boldsymbol{v}} = \frac{(2, -4,3)}{\sqrt{2^2 + (-4)^2 + 3^2}} = \left(\frac{2}{\sqrt{29}}, \frac{-4}{\sqrt{29}}, \frac{3}{\sqrt{29}}\right) \approx (0{,}3714,\ -0{,}7428,\ 0{,}5571)$$

W kodzie wektor jednostkowy możemy wyznaczyć w następujący sposób:

```
>>> v = np.array((2, -4, 3))
>>> u = v / np.sqrt((v*v).sum())
>>> print(u)
[ 0.37139068 -0.74278135 0.55708601 ]
```


Kup ksi k Pole ksi k

W powyższym kodzie korzystam z tego, że kwadraty poszczególnych elementów wektora $\boldsymbol{v}$ możemy wyznaczyć, mnożąc go przez niego samego (mnożenie na elementach). Po wyznaczeniu kwadratów za pomocą `sum` obliczam ich sumę i otrzymuję kwadrat odległości.

Transpozycja wektora

Wspomniałem wcześniej, że wektory wierszowe można traktować jako macierze o wymiarach $1 \times n$, podczas gdy wektory kolumnowe to macierze $n \times 1$. Zamianę wektora wierszowego na wektor kolumnowy i *vice versa* nazywamy **transpozycją**. W rozdziale 6. przekonasz się, że transponować możemy także macierze. Transpozycję wektora $\boldsymbol{y}$ oznaczamy $\boldsymbol{y}^\top$. A zatem mamy

$$\boldsymbol{x} = \begin{bmatrix} x_0 & x_1 & x_2 \end{bmatrix}$$

$$\boldsymbol{x}^\top = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}$$

$$y = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \end{bmatrix}$$

$$\boldsymbol{y}^\top = \begin{bmatrix} y_0 & y_1 & y_2 \end{bmatrix}$$

$$(\boldsymbol{z}^\top)^\top = \boldsymbol{z}$$

Oczywiście transpozycja nie ogranicza się do wektorów trzyelementowych.

W kodzie transpozycję wektora można przeprowadzić na kilka sposobów. Jak widziałeś we wcześniejszej części rozdziału, do zamiany wektora w macierz $1 \times n$ lub $n \times 1$ możemy wykorzystać funkcję `reshape`. Można też wywołać na wektorze metodę `transpose` lub przy zachowaniu pewnej dozy ostrożności użyć skrótu `T`. Spójrz na przykłady wszystkich tych podejść. Zacznę od zdefiniowania wektora NumPy. Spójrz, w jaki sposób `reshape` zamienia zwykły trzyelementowy wektor w wektor kolumnowy o wymiarach 3×1 i wektor wierszowy o wymiarach 1×3:

```
>>> v = np.array([1,2,3])
>>> print(v)
[1 2 3]
>>> print(v.reshape((3,1)))
[[1]
 [2]
 [3]]
>>> print(v.reshape((1,3)))
[[1 2 3]]
```

Zwróć uwagę na różnicę pomiędzy wyjściem z `print(v)` a wynikiem wyświetlonym po wywołaniu `reshape((1,3))`. W drugim przypadku dane wyjściowe są otoczone dodatkową parą nawiasów wskazującą, że pierwszy wymiar ma długość 1.

Kup ksi k Pole ksi k

Teraz zastosuję operację transpozycji na `v`:

```
>>> print(v.transpose())
[1 2 3]
>>> print(v.T)
[1 2 3]
```

Jak widzisz, wywołanie `transpose` lub `T` w żaden sposób nie wpływa na `v`. Jest tak, ponieważ kształt `v` to po prostu `3`, a nie `(1,3)` lub `(3,1)`. Po jawnej zamianie `v` w macierz o wymiarach 1×3 `transpose` i `T` dają pożądany efekt:

```
>>> v = v.reshape((1,3))
>>> print(v.transpose())
[[1]
 [2]
 [3]]
>>> print(v.T)
[[1]
 [2]
 [3]]
```

Na powyższym listingu `v` zmienia się z wektora wierszowego w kolumnowy, zgodnie z naszymi oczekiwaniami. Lekcją z tego przykłady jest konieczność zachowania ostrożności co do rzeczywistej wymiarowości wektora w NumPy. W większości przypadków możemy się nią nie przejmować, ale czasami musimy wyrażać się jasno i zwracać uwagę na rozróżnienie pomiędzy zwykłymi wektorami, wektorami wierszowymi i wektorami kolumnowymi.

Iloczyn wewnętrzny

Być może najpopularniejszą operacją wektorową jest **iloczyn wewnętrzny**, czyli inaczej **iloczyn skalarny**. Iloczyn skalarny pomiędzy dwoma wektorami definiuje się w następujący sposób:

$$\begin{aligned} \boldsymbol{a} \bullet \boldsymbol{b} &= \langle \boldsymbol{a}, \boldsymbol{b} \rangle = \boldsymbol{a}^\top \boldsymbol{b} \\ &= \sum_{k=0}^{n-1} a_k b_k \qquad (5.4) \\ &= \| \boldsymbol{a} \| \| \boldsymbol{b} \| \cos \theta \qquad (5.5) \end{aligned}$$

W powyższym równaniu θ to kąt pomiędzy dwoma wektorami (jeśli mają one interpretację geometryczną). Wynikiem iloczynu skalarnego jest skalar. Zapis $\langle \boldsymbol{a}, \boldsymbol{b} \rangle$ jest dość często spotykany, chociaż w literaturze poświęconej uczeniu głębokiemu częściej stosuje się zapis $\boldsymbol{a} \bullet \boldsymbol{b}$. Zapis z użyciem mnożenia macierzy ($\boldsymbol{a}^\top \boldsymbol{b}$) mówi nam, jak obliczyć iloczyn skalarny, ale z jego wyjaśnieniem poczekam do omówienia mnożenia macierzy. Na razie wszystko, co powinieneś wiedzieć, jest ukryte we wzorze z sumą — iloczyn skalarny dwóch n-elementowych wektorów jest sumą n iloczynów odpowiadających sobie składowych.

Kup ksi k Pole ksi k

Iloczyn skalarny wektora z nim samym to kwadrat jego długości:

$$\boldsymbol{a}\bullet\boldsymbol{a} = \| \boldsymbol{a} \|^2$$

Iloczyn wewnętrzny jest przemienny:

$$\boldsymbol{a}\bullet\boldsymbol{b} = \boldsymbol{b}\bullet\boldsymbol{a}$$

oraz rozdzielny względem dodawania:

$$\boldsymbol{a}\bullet(\boldsymbol{b} + \boldsymbol{c}) = \boldsymbol{a}\bullet\boldsymbol{b} + \boldsymbol{a}\bullet\boldsymbol{c}$$

ale nie jest łączny, ponieważ wynikiem pierwszego iloczynu skalarnego jest skalar, a nie wektor, a mnożenie wektora przez skalar to nie iloczyn skalarny.

Zauważ też, że iloczyn skalarny jest równy zero, gdy kąt między wektorami wynosi 90 stopni. Wynika to z tego, że dla tej miary kąta cos θ jest równy zero (równanie 5.5). Zerowy iloczyn skalarny oznacza, że wektory są do siebie prostopadłe lub **ortogonalne**.

Spójrzmy na kilka przykładów iloczynów skalarnych. Zacznij od bezpośredniej implementacji równania 5.4:

```
>>> a = np.array([1,2,3,4])
>>> b = np.array([5,6,7,8])
>>> def inner(a,b):
...   s = 0.0
...   for i in range(len(a)):
...     s += a[i]*b[i]
...   return s ...
...
>>> inner(a,b)
70.0
```

Ponieważ wiemy, że `a` i `b` są tablicami NumPy, powyższy kod możemy zapisać w bardziej wydajny sposób:

```
>>> (a*b).sum()
70
```

Możemy też, co prawdopodobnie będzie najwydajniejszym rozwiązaniem, pozwolić NumPy wykonać całą robotę za nas i po prostu wywołać `np.dot`:

```
>>> np.dot(a,b)
70
```

Wywołanie `np.dot` pojawia się dość często w kodach algorytmów uczenia głębokiego. Jak się wkrótce przekonasz, funkcja ta pozwala na wiele więcej niż tylko obliczenie iloczynu skalarnego.

Z równania 5.5 wiemy, że kąt między dwoma wektorami to

$$\theta = \cos^{-1} \frac{\boldsymbol{a} \bullet \boldsymbol{b}}{\| \boldsymbol{a} \| \| \boldsymbol{b} \|}$$

W kodzie obliczenia te można zaimplementować w następujący sposób:

```
>>> A = np.sqrt(np.dot(a,a))
>>> B = np.sqrt(np.dot(b,b))
>>> t = np.arccos(np.dot(a,b)/(A*B))
>>> t*(180/np.pi)
14.335170291600924
```

Wynik działania powyższego kodu mówi nam, że kąt pomiędzy $\boldsymbol{a}$ i $\boldsymbol{b}$ wynosi po przekonwertowaniu wartości `t` z radianów na stopnie około 14°.

W przypadku przestrzeni trójwymiarowej iloczyn skalarny jest równy zero dla wektorów ortogonalnych, co oznacza, że kąt pomiędzy nimi wynosi 90°:

```
>>> a = np.array([1,0,0])
>>> b = np.array([0,1,0])
>>> np.dot(a,b)
0
>>> t = np.arccos(0)
>>> t*(180/np.pi)
90.0
```

Powyższe wyniki mają sens, ponieważ `a` jest wektorem jednostkowym leżącym na osi x, a `b` jest wektorem jednostkowym leżącym na osi y. Wiemy zatem, że między nimi jest kąt prosty.

Po zapoznaniu się z iloczynem skalarnym spójrz, jak możemy go wykorzystać do rzutowania jednego wektora na drugi.

Rzut

Rzut jednego wektora na drugi daje nam informację, jaka część pierwszego wektora jest wektorem o kierunku drugiego. Rzut $\boldsymbol{a}$ na $\boldsymbol{b}$ to

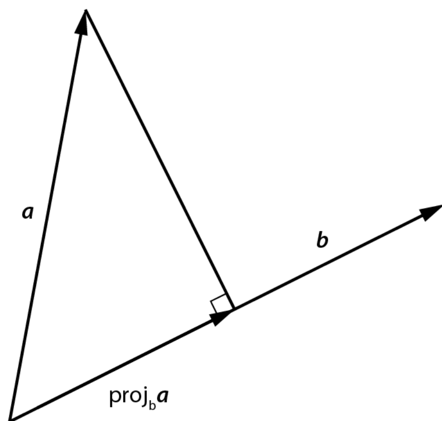
$$\mathbf{proj}_{b}\ \boldsymbol{a} = \frac{\boldsymbol{a} \bullet \boldsymbol{b}}{\| \boldsymbol{b} \|^2} \boldsymbol{b}$$

Na rysunku 5.1 pokazano wizualizację rzutu na przykładzie wektorów 2D.

Rzut polega na wyznaczeniu składowej $\boldsymbol{a}$ wskazującej w kierunku $\boldsymbol{b}$. Zauważ, że rzut $\boldsymbol{a}$ na $\boldsymbol{b}$ nie jest tym samym co rzut $\boldsymbol{b}$ na $\boldsymbol{a}$.

Ponieważ w liczniku pojawia się iloczyn skalarny, rzut wektora na inny, ortogonalny do niego wektor jest równy zero. Żadna składowa pierwszego wektora nie jest w takim przypadku zwrócona w kierunku drugiego wektora. Raz jeszcze powróćmy do osi x i y. Powodem, dla którego korzystamy ze współrzędnych kartezjańskich, jest to, że w układzie tym jego dwie lub trzy (w przestrzeni 3D) osie są wzajemnie ortogonalne, tj. żadna część żadnej z nich nie jest zwrócona w stronę pozostałych. Dzięki temu możemy określić

Kup ksi k Pole ksi k



Rysunek 5.1. Rzut a na b w przestrzeni 2D

położenie dowolnego punktu oraz wektor łączący ten punkt z początkiem układu współrzędnych za pomocą składowych leżących na tych osiach. Rozkład obiektu na wzajemnie ortogonalne składowe pokażę Ci w rozdziale 6. przy okazji omawiania wektorów własnych i metody PCA.

W kodzie wykonanie rzutu jest proste:

```
>>> a = np.array([1,1])
>>> b = np.array([1,0])
>>> p = (np.dot(a,b)/np.dot(b,b))*b
>>> print(p)
[1. 0.]
>>> c = np.array([-1,1])
>>> p = (np.dot(c,b)/np.dot(b,b))*b
>>> print(p)
[-1. -0.]
```

W pierwszym przykładzie $\boldsymbol{a}$ wskazuje o 45° w lewo w kierunku przeciwnym do ruchu wskazówek zegara od osi x, natomiast $\boldsymbol{b}$ wskazuje w kierunku osi $\boldsymbol{x}$. Oczekiwalibyśmy, że rzut $\boldsymbol{a}$ będzie przebiegał wzdłuż osi $\boldsymbol{x}$ i tak też jest (p). W drugim przykładzie $\boldsymbol{c}$ wskazuje o $135° = 90° + 45°$ w lewo w kierunku przeciwnym do ruchu wskazówek zegara od osi x. A zatem spodziewamy się, że składowa $\boldsymbol{c}$ w kierunku $\boldsymbol{b}$ będzie przebiegała wzdłuż osi x w kierunku przeciwnym do $\boldsymbol{b}$. Tak też jest.

Uwaga

Rzut c *na* b *dał składową względem osi* y *równą* -0. *Obecność minusa wynika ze sposobu reprezentacji liczb zmiennoprzecinkowych w standardzie IEEE 754. Choć mantysa wynosi zero, nadal istnieje możliwość ustalenia znaku, co od czasu do czasu prowadzi do minus zera. Szczegółowe wyjaśnienie sposobu zapisu liczb na komputerze, w tym opis zapisu zmiennoprzecinkowego, znajdziesz w napisanej przeze mnie książce* Numbers and Computers *(Springer-Verlag, 2017).*

Przejdę teraz do omówienia iloczynu zewnętrznego dwóch wektorów.

Kup książkę Poleć książkę

Iloczyn zewnętrzny

Iloczyn wewnętrzny (skalarny) dwóch wektorów daje skalar, natomiast wynikiem **iloczynu zewnętrznego** jest macierz. W przeciwieństwie do iloczynu skalarnego w iloczynie zewnętrznym nie wymagamy, aby oba wektory miały tę samą liczbę elementów. Iloczyn zewnętrzny m-elementowego wektora $\boldsymbol{a}$ z n-elementowym wektorem $\boldsymbol{b}$ to macierz utworzona poprzez pomnożenie każdego elementu $\boldsymbol{a}$ przez każdy element $\boldsymbol{b}$:

$$\boldsymbol{a} \otimes \boldsymbol{b} = \boldsymbol{a}\boldsymbol{b}^\top = \begin{bmatrix} a_0 b_0 & a_0 b_1 & a_0 b_2 & \cdots & a_0 b_{n-1} \\ a_1 b_0 & a_1 b_1 & a_1 b_2 & \cdots & a_1 b_{n-1} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ a_{m-1} b_0 & a_{m-1} b_1 & a_{m-1} b_2 & \cdots & a_{m-1} b_{n-1} \end{bmatrix}$$

Formuła $\boldsymbol{a}\boldsymbol{b}^\top$ opisuje sposób na wyznaczenie iloczynu zewnętrznego za pomocą mnożenia macierzy. Zauważ, że formuła ta różni się od formuły opisującej iloczyn skalarny ($\boldsymbol{a}^\top\boldsymbol{b}$) i że zakłada się w niej, że $\boldsymbol{a}$ i $\boldsymbol{b}$ są wektorami kolumnowymi. Nie istnieje żaden powszechnie akceptowany symbol do oznaczenia iloczynu zewnętrznego (głównie dlatego, że można go zapisać jako mnożenie macierzy i że nie jest on tak powszechny jak iloczyn skalarny), ale w kontekście, w którym przedstawia się go jako operację dwuargumentową, dość często stosuje się symbol $\otimes$.

W NumPy dostępna jest funkcja `np.outer`:

```
>>> a = np.array([1,2,3,4])
>>> b = np.array([5,6,7,8])
>>> np.dot(a,b)
70
>>> np.outer(a,b)
array([[ 5,  6,  7,  8],
       [10, 12, 14, 16],
       [15, 18, 21, 24],
       [20, 24, 28, 32]])
```

Wektory `a` i `b` to te same wektory co w przykładach z iloczynem skalarnym. Zgodnie z oczekiwaniami `np.dot` zwraca iloczyn skalarny $\boldsymbol{a} \bullet \boldsymbol{b}$. Natomiast funkcja `np.outer` zwraca macierz 4×4, w której kolejne wiersze to wyniki mnożenia `b` przez kolejne elementy wektora `a`; najpierw przez `1`, potem przez `2`, potem przez `3` i na końcu przez `4`. A zatem pomnożyliśmy każdy element wektora `b` przez każdy kolejny element wektora `a`. Wynikiem jest macierz 4×4, ponieważ zarówno `a`, jak i `b` mają po cztery elementy.

Zdolność iloczynu zewnętrznego do tworzenia wszystkich kombinacji danych wejściowych znajduje zastosowanie w uczeniu głębokim w zagadnieniach takich jak *neural collaborative filtering* i *visual questions answering*. Zadania te są realizowane przez zaawansowane sieci neuronowe, które wydają rekomendacje lub odpowiadają na pytania tekstowe dotyczące zawartości obrazów. Iloczyny zewnętrzne mieszają w nich dwa różne wektory osadzające. **Osadzenia** (ang. *embeddings*) to wektory generowane przez wcześniejsze warstwy sieci neuronowej, na przykład przez przedostatnią w pełni połączoną warstwę znajdującą się przed wyjściem warstwy *softmax* w tradycyjnej konwolucyjnej sieci neuronowej. Warstwę generującą osadzenia traktuje się zazwyczaj jako warstwę, która nauczyła się nowej reprezentacji wejścia. Można ją traktować jako odwzorowanie złożonych danych wejściowych, takich jak obrazy, na zredukowaną przestrzeń zawierającą od kilkuset do kilku tysięcy wymiarów.

Kup ksi k Pole ksi k

ILOCZYN KARTEZJAŃSKI

Istnieje bezpośrednia analogia pomiędzy iloczynem zewnętrznym dwóch wektorów i iloczynem kartezjańskim dwóch zbiorów A i B. Iloczyn kartezjański jest nowym zbiorem, którego elementami są wszystkie możliwe pary elementów z A i B. A zatem jeśli $A = \{1,2,3,4\}$ i $B = \{5,6,7,8\}$, to iloczyn kartezjański tych zbiorów możemy zapisać jako

$$\begin{aligned} A \times B &= \{(a, b) \mid a \in A \text{ oraz } b \in B\} \\ &= \begin{pmatrix} (1,5) & (1,6) & (1,7) & (1,8) \\ (2,5) & (2,6) & (2,7) & (2,8) \\ (3,5) & (3,6) & (3,7) & (3,8) \\ (4,5) & (4,6) & (4,7) & (4,8) \end{pmatrix} \end{aligned}$$

Jeśli zastąpimy każdą parę iloczynem jej elementów, to otrzymamy iloczyn zewnętrzny pokrywający się z wynikiem zwróconym przez `np.outer`. Do oznaczania iloczynu kartezjańskiego dwóch zbiorów często wykorzystuje się symbol ×.

Iloczyn wektorowy

Ostatnim pozostałym do omówienia operatorem wektor-wektor jest **iloczyn wektorowy**. Operator ten jest zdefiniowany tylko dla przestrzeni 3D ($\mathbb{R}^3$). Iloczynem wektorowym $\boldsymbol{a}$ i $\boldsymbol{b}$ jest nowy wektor prostopadły do płaszczyzny rozpinanej przez $\boldsymbol{a}$ i $\boldsymbol{b}$. Zauważ, że nie oznacza to, że $\boldsymbol{a}$ i $\boldsymbol{b}$ są do siebie prostopadłe. Iloczyn wektorowy definiuje się jako

$$\begin{aligned} \boldsymbol{a} \times \boldsymbol{b} &= \| \boldsymbol{a} \| \| \boldsymbol{b} \| \sin(\theta)\, \hat{\boldsymbol{n}} \\ &= (a_1 b_2 - a_2 b_1, a_2 b_0 - a_0 b_2, a_0 b_1 - a_1 b_0) \end{aligned} \tag{5.6}$$

gdzie $\hat{\boldsymbol{n}}$ jest wektorem jednostkowym, a θ kątem pomiędzy $\boldsymbol{a}$ i $\boldsymbol{b}$. Kierunek $\hat{\boldsymbol{n}}$ jest określony przez **regułę prawej ręki**. Skieruj palec wskazujący prawej ręki w kierunku $\boldsymbol{a}$, a palec środkowy w kierunku $\boldsymbol{b}$. Po ich skierowaniu kciuk będzie wskazywał w kierunku $\hat{\boldsymbol{n}}$. Równanie 5.6 zawiera wzory na poszczególne składowe wynikowego wektora.

W NumPy do obliczania iloczynu wektorowego służy funkcja `np.cross`:

```
>>> a = np.array([1,0,0])
>>> b = np.array([0,1,0])
>>> print(np.cross(a,b))
[0 0 1]
>>> c = np.array([1,1,0])
>>> print(np.cross(a,c))
[0 0 1]
```

W pierwszym przykładzie $\boldsymbol{a}$ wskazuje w kierunku osi x, a b w kierunku osi y. Dlatego oczekujemy, że iloczyn wektorowy będzie prostopadły do tych osi i będzie to wektor wskazujący wzdłuż osi z. Drugi przykład pokazuje, że nie ma znaczenia, czy $\boldsymbol{a}$ i $\boldsymbol{b}$ są do siebie prostopadłe. W drugim przypadku $\boldsymbol{c}$ znajduje się pod kątem 45° w stosunku do osi x, ale $\boldsymbol{a}$ i $\boldsymbol{c}$ nadal leżą w płaszczyźnie xy. Dlatego iloczyn wektorowy nadal leży wzdłuż osi z.

Kup ksi k Pole ksi k

Definicja iloczynu wektorowego zawiera sin θ, a definicja iloczynu wewnętrznego cos θ. Iloczyn wewnętrzny jest równy zero, gdy dwa wektory są do siebie ortogonalne. Z drugiej strony iloczyn wektorowy jest równy zero, gdy dwa wektory wskazują w tym samym kierunku. Iloczyn wektorowy osiąga maksimum, gdy wektory są do siebie prostopadłe. W drugim z powyższych przykładów NumPy daje wektor o składowej równej 1, ponieważ długość $\boldsymbol{c}$ jest równa $\sqrt{2}$, a $\sin\ 45^\circ = \sqrt{2}/2 = 1/\sqrt{2}$. W rezultacie $\sqrt{2}$ ulega skróceniu i otrzymujemy w wyniku 1, ponieważ $\boldsymbol{a}$ jest wektorem jednostkowym.

Iloczyn wektorowy jest powszechnie stosowany w fizyce i innych naukach, natomiast ze względu na jego ograniczenie jedynie do przestrzeni trójwymiarowej nie spotyka się go zbyt często w uczeniu głębokim. Niemniej powinieneś się z nim zapoznać, jeśli masz zamiar zajrzeć do literatury poświęconej uczeniu głębokiemu.

Na tym kończy się nasze spojrzenie na operacje typu wektor-wektor. Pozostawię już jednowymiarowy świat i przejdę do omówienia najważniejszej operacji w całym uczeniu głębokim, czyli do mnożenia macierzy.

Mnożenie macierzy

W poprzednim punkcie pokazałem Ci następujące sposoby na pomnożenie dwóch wektorów: iloczyn Hadamarda, iloczyn skalarny (wewnętrzny), iloczyn zewnętrzny i iloczyn wektorowy. W tym punkcie zajmiemy się mnożeniem macierzy. Pamiętaj, że wektory wierszowe i kolumnowe możemy traktować jako macierze posiadające jeden wiersz lub kolumnę.

Własności mnożenia macierzy

Wkrótce zdefiniuję mnożenie macierzy, ale zanim to zrobię, przyjrzymy się jego właściwościom. Niech $\boldsymbol{A}, \boldsymbol{B}$ i $\boldsymbol{C}$ będą macierzami. Zgodnie z powszechnie stosowaną w algebrze konwencją zapisanie koło siebie dwóch symboli bez żadnego symbolu pomiędzy nimi oznacza ich iloczyn. Dla mnożenia macierzowego prawdą jest, że:

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

co oznacza, że mnożenie macierzy jest łączne. Mnożenie macierzy jest też rozłączne względem dodawania:

$$\boldsymbol{A}(\boldsymbol{B} + \boldsymbol{C}) = \boldsymbol{AB} + \boldsymbol{AC} \quad (5.7)$$

$$(\boldsymbol{A} + \boldsymbol{B})\boldsymbol{C} = \boldsymbol{AC} + \boldsymbol{BC} \quad (5.8)$$

Natomiast w ogólnym przypadku *nie* jest to działanie przemienne:

$$\boldsymbol{AB} \neq \boldsymbol{BA}$$

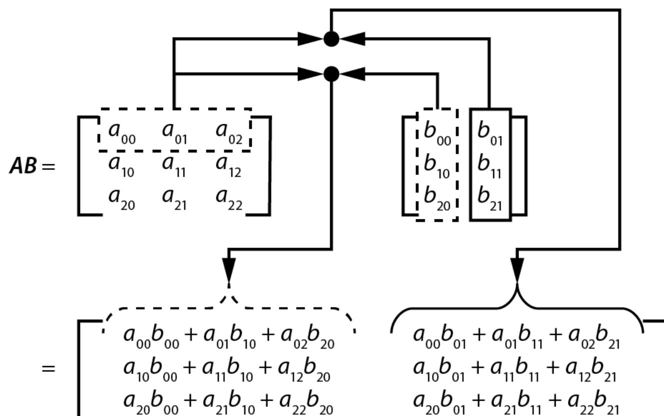
Równanie 5.8 pokazuje nam, że prawostronne pomnożenie sumy macierzy przez macierz daje inny wynik niż lewostronne mnożenie tej samej sumy widoczne w równaniu 5.7. Z tego powodu musiałem zamieścić zarówno równanie 5.7, jak i 5.8. Mnożenie macierzowe można wykonywać od lewej lub prawej strony, natomiast wyniki tych mnożeń zazwyczaj różnią się od siebie.

Kup ksi k Pole ksi k

Jak pomnożyć ze sobą dwie macierze

Przed wyznaczeniem iloczynu $\boldsymbol{AB}$, w którym $\boldsymbol{A}$ musi znajdować się po lewej stronie od $\boldsymbol{B}$, musimy sprawdzić zgodność wymiarów. Pomnożenie dwóch macierzy jest możliwe tylko wtedy, gdy liczba kolumn w $\boldsymbol{A}$ jest równa liczbie wierszy w $\boldsymbol{B}$. A zatem jeśli $\boldsymbol{A}$ jest macierzą $n \times m$, a $\boldsymbol{B}$ jest macierzą $m \times k$, to możemy obliczyć iloczyn $\boldsymbol{AB}$, a jego wynikiem będzie nowa macierz o wymiarach $n \times k$.

Aby obliczyć iloczyn, obliczamy kolejne iloczyny skalarne pomiędzy wektorami wierszowymi z $\boldsymbol{A}$ i wektorami kolumnowymi z $\boldsymbol{B}$. Na rysunku 5.2 pokazano proces mnożenia macierzy $\boldsymbol{A}$ o wymiarach 3×3 przez macierz $\boldsymbol{B}$ o wymiarach 3×2.



Rysunek 5.2. Mnożenie macierzy o wymiarach 3×3 *przez macierz o wymiarach* 3×2

Pierwszy wiersz macierzy wynikowej pokazanej na rysunku 5.2 powstaje poprzez obliczenie iloczynów skalarnych pierwszego wiersza macierzy $\boldsymbol{A}$ z kolejnymi kolumnami macierzy $\boldsymbol{B}$. Pierwszy element macierzy wyjściowej to wynik iloczynu skalarnego pierwszego wiersza macierzy $\boldsymbol{A}$ z pierwszą kolumną $\boldsymbol{B}$. Drugi i ostatni element pierwszego wiersza macierzy wyjściowej to wynik iloczynu skalarnego pierwszego wiersza macierzy $\boldsymbol{A}$ z drugą kolumną macierzy $\boldsymbol{B}$.

Spójrz na przykład mnożenia macierzy z rysunku 5.2, tym razem z konkretnymi wartościami:

$$\boldsymbol{AB} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} 11 & 22 \\ 33 & 44 \\ 55 & 66 \end{bmatrix}$$

$$= \begin{bmatrix} (1)11 + (2)33 + (3)55 & (1)22 + (2)44 + (3)66 \\ (4)11 + (5)33 + (6)55 & (4)22 + (5)44 + (6)66 \\ (7)11 + (8)33 + (9)55 & (7)22 + (8)44 + (9)66 \end{bmatrix}$$

$$= \begin{bmatrix} 242 & 308 \\ 539 & 704 \\ 836 & 1100 \end{bmatrix}$$

Zauważ, że $\boldsymbol{AB}$ jest zdefiniowane, a $\boldsymbol{BA}$ nie, ponieważ nie możemy pomnożyć macierzy o wymiarach 3×2 przez macierz o wymiarach 3×3. Liczba kolumn w $\boldsymbol{B}$ musi być taka sama jak liczba wierszy w $\boldsymbol{A}$.

Innym sposobem myślenia o mnożeniu macierzy jest zastanowienie się, co składa się na każdy z elementów wynikowej macierzy Na przykład, jeśli $\boldsymbol{A}$ ma wymiary $n \times m$, a $\boldsymbol{B}$ to macierz $m \times p$, wiemy, że wynikiem ich mnożenia będzie macierz $\boldsymbol{C}$ o wymiarach $n \times p$. Elementy macierzy wynikowej możemy wyznaczyć za pomocą następującego wzoru:

$$c_{ij} = \sum_{k=0}^{m-1} a_{ik} b_{kj} \tag{5.9}$$

dla $i = 0, \dots, n-1$ oraz $j = 0, \dots, p-1$. Np. aby znaleźć element c_{21} z powyższego przykładu, musimy obliczyć sumę $a_{20}b_{01} + a_{21}b_{11} + a_{22}b_{21}$. Formuła ta powstaje po wstawieniu do równania 5.9 $i = 2, j = 1$ i $k = 0,1,2$.

Równanie 5.9 mówi nam, jak znaleźć pojedynczy element macierzy wyjściowej. Aby znaleźć całą macierz, musimy za pomocą pętli przejść przez wszystkie wartości i oraz j. Prowadzi to do prostej implementacji mnożenia macierzy:

```
def matrixmul(A,B):
    I,K = A.shape
    J = B.shape[1]
    C = np.zeros((I,J), dtype=A.dtype)
    for i in range(I):
        for j in range(J):
            for k in range(K):
                C[i,j] += A[i,k]*B[k,j]
    return C
```

W powyższej implementacji zakładam, że $\boldsymbol{A}$ i $\boldsymbol{B}$ mają odpowiednie wymiary. Wyznaczam liczbę wierszy (`I`) i kolumn (`J`) macierzy wyjściowej $\boldsymbol{C}$ i wykorzystuję je jako ograniczenia pętli przechodzących przez elementy $\boldsymbol{C}$. Następnie tworzę macierz wyjściową `C` i ustawiam jej typ danych na typ danych macierzy `A`. W następnych wierszach znajduje się potrójnie zagnieżdżona pętla. Pętla z iteratorem `i` przechodzi przez wszystkie wiersze wynikowej macierzy. Druga pętla, z iteratorem `j`, odwiedza wszystkie kolumny aktualnie przetwarzanego wiersza, a najbardziej wewnętrzna pętla po `k` sumuje iloczyny odpowiednich wartości `A` i `B` zgodnie z równaniem 5.9. Po zakończeniu działania wszystkich pętli zwracam macierz `C` zawierającą wynik mnożenia.

Funkcja `matrixmul` działa i poprawnie mnoży macierze, natomiast jej implementacja jest dość naiwna. Istnieją bardziej zaawansowane algorytmy mnożenia macierzy, a także metody optymalizacji skompilowanych wersji naiwnych implementacji. Jak przekonasz się w dalszej części rozdziału, NumPy zawiera implementację mnożenia macierzy, która korzysta z procedur pochodzących z wysoce zoptymalizowanych kompilowanych bibliotek. Procedura ta jest znacznie bardziej wydajna niż powyższy kod.

Kup książkę Poleć książkę

Notacja macierzowa dla iloczynów wewnętrznych i zewnętrznych

Masz już niezbędną wiedzę, aby zrozumieć, dlaczego $\boldsymbol{a}^\top \boldsymbol{b}$ to iloczyn skalarny, a $\boldsymbol{a}\boldsymbol{b}^\top$ to iloczyn zewnętrzny dwóch wektorów. W pierwszym przypadku mnożymy wektor wierszowy o wymiarach $1 \times n$ (ze względu na transpozycję) z wektorem kolumnowym o wymiarach $n \times 1$. Algorytm mnożenia macierzy mówi nam, że iloczyn wektora wierszowego z wektorem kolumnowym daje macierz o wymiarach 1×1, czyli skalar. Zauważ, że zarówno $\boldsymbol{a}$, jak i $\boldsymbol{b}$ muszą zawierać po n-elementów.

W przypadku iloczynu zewnętrznego po lewej mamy wektor kolumnowy o wymiarach $n \times 1$, a po prawej wektor wierszowy o wymiarach $1 \times m$. Wiemy więc, że wynikowa macierz będzie macierzą $n \times m$. Jeśli $m = n$, to otrzymamy macierz $n \times n$. Macierz o tej samej liczbie wierszy i kolumn nazywamy **macierzą kwadratową**. Macierze te mają specjalne własności, z których część pokażę Ci w rozdziale 6.

Aby znaleźć iloczyn zewnętrzny dwóch wektorów za pomocą mnożenia macierzy, należy pomnożyć kolejne wiersze $\boldsymbol{a}$ przez każdą kolumnę wektora $\boldsymbol{b}$, który jest **wektorem wierszowym**:

$$\begin{aligned}\boldsymbol{a}\boldsymbol{b}^\top &= \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} \begin{bmatrix} b_0 & b_1 & b_2 \end{bmatrix} \\ &= \begin{bmatrix} a_0 b_0 & a_0 b_1 & a_0 b_2 \\ a_1 b_0 & a_1 b_1 & a_1 b_2 \\ a_2 b_0 & a_2 b_1 & a_2 b_2 \end{bmatrix}\end{aligned}$$

Każda kolumna $\boldsymbol{b}^\top$ to skalar, który mnożymy z każdym wierszem $\boldsymbol{a}$, tworząc w ten sposób każdy możliwy iloczyn pomiędzy elementami tych dwóch wektorów.

Pokazałem Ci już, jak wygląda ręczne mnożenie macierzy. Czas przyjrzeć się temu, jak robi to NumPy.

Mnożenie macierzy w NumPy

NumPy zawiera dwie różne funkcje, które możemy wykorzystać do mnożenia macierzy. Pierwsza, widziana już przez nas funkcja, to `np.dot`. Funkcję tę wykorzystywaliśmy do tej pory jedynie do obliczania iloczynu skalarnego wektorów. Drugą funkcją jest natomiast `np.matmul`. Funkcja ta jest wywoływana również przez operator `@` dostępny od Pythona 3.5. Obie funkcje mnożą macierze zgodnie z naszymi oczekiwaniami. Natomiast NumPy czasami traktuje jednowymiarowe tablice inaczej niż wektory wierszowe lub kolumnowe.

Za pomocą atrybutu `shape` możemy sprawdzić, czy nasz wektor jest jednowymiarową tablicą NumPy, wektorem wierszowym, czy wektorem kolumnowym (listing 5.1).

Listing 5.1. Wektory w NumPy

```
>>> av = np.array([1,2,3])
>>> ar = np.array([[1,2,3]])
>>> ac = np.array([[1],[2],[3]])
>>> av.shape
(3,)
>>> ar.shape
```


Kup książkę Poleć książkę

```
(1, 3)
>>> ac.shape
(3, 1)
```

Jak widzisz, jednowymiarowa, trzyelementowa tablica `av` ma inny kształt niż trzyelementowy wektor wierszowy `ar` oraz trzyelementowy wektor kolumnowy `ac`. Natomiast każda z tych tablic zawiera te same trzy liczby całkowite: `1`, `2` i `3`.

Przeprowadźmy eksperyment, który pomoże nam zrozumieć, w jaki sposób zaimplementowano mnożenie macierzy w NumPy. W testach wykorzystam funkcję `np.dot`, ale jeśli skorzystasz z `np.matmul` lub operatora `@`, to otrzymasz te same wyniki. Do testów potrzebny nam będzie zbiór wektorów i macierzy. Przetestujemy działanie funkcji `np.dot` na różnych kombinacjach wektorów i macierzy z tego zbioru i przyjrzymy się wynikom. Niektóre wywołania mogą zwrócić błąd w przypadku, w którym wynik dla danej kombinacji argumentów będzie niezdefiniowany.

Na poniższym listingu tworzę tablice, wektory i macierze niezbędne do przeprowadzenia testów:

```
a1 = np.array([1,2,3])
ar = np.array([[1,2,3]])
ac = np.array([[1],[2],[3]])
b1 = np.array([1,2,3])
br = np.array([[1,2,3]])
bc = np.array([[1],[2],[3]])
A = np.array([[1,2,3],[4,5,6],[7,8,9]])
B = np.array([[9,8,7],[6,5,4],[3,2,1]])
```

Jeśli masz w pamięci wyniki z listingu 5.1, to powinieneś bez problemu ustalić kształty zdefiniowanych powyżej obiektów. Na powyższym listingu zdefiniowałem też dwie macierze A oraz B o wymiarach 3×3.

Zdefiniuję też funkcję pomocniczą, która opakuje wywołanie `np.dot` i przechwyci ewentualne błędy:

```
def dot(a,b):
    try:
        return np.dot(a,b)
    except:
        return "błąd"
```

Funkcja ta wywołuje `np.dot` i zwraca słowo `błąd`, jeśli wywołanie zakończy się niepowodzeniem. W tabeli 5.1 znajdziesz wyniki dla różnych kombinacji zdefiniowanych powyżej obiektów.

Tabela 5.1 pokazuje, w jaki sposób NumPy czasami traktuje jednowymiarowe tablice inaczej niż wektory wierszowe lub kolumnowe. Porównaj wynik obliczeń dla `a1,A` z wynikami mnożeń `ar,A` i `A,ac`. Wynik dla `A,ac` jest tym, który spodziewalibyśmy się zobaczyć na zajęciach z matematyki, czyli wynikiem lewostronnego mnożenia wektora kolumnowego $\boldsymbol{a_c}$ przez $\boldsymbol{A}$.

Kup ksi k Pole ksi k

Tabela 5.1. Wyniki zastosowania funkcji `np.dot` lub `np.matmul` na argumentach różnego typu

Argumenty	Wynik zastosowania `np.dot` lub `np.matmul`
`a1, b1`	14 (skalar)
`a1, br`	błąd
`a1, bc`	$[14]$ (wektor jednoelementowy)
`ar, b1`	$[14]$ (wektor jednoelementowy)
`ar, br`	błąd
`ar, bc`	$[14]$ (macierz 1×1)
`ac, b1`	błąd
`ac, br`	$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{bmatrix}$ (iloczyn zewnętrzny)
`ac, bc`	błąd
`A, a1`	$[14 \quad 32 \quad 50]$ (wektor trzyelementowy)
`A, ar`	błąd
`A, ac`	$\begin{bmatrix} 14 \\ 32 \\ 50 \end{bmatrix}$
`a1, A`	$[30 \quad 36 \quad 42]$ (wektor trzyelementowy)
`ar, A`	$[30 \quad 36 \quad 42]$ (macierz 1×3)
`ac, A`	błąd
`A, B`	$\begin{bmatrix} 30 & 24 & 18 \\ 84 & 69 & 54 \\ 138 & 114 & 90 \end{bmatrix}$

Czy istnieje jakaś realna różnica pomiędzy `np.dot` i `np.matmul`? Pewna różnica jest. W przypadku jedno- i dwuwymiarowych tablic nie ma żadnych różnic. Funkcje te różnią się od siebie sposobem obsługi tablic o więcej niż dwóch wymiarach, ale w tej książce nie będę poruszał tego zagadnienia. Ponadto w przypadku `np.dot` jednym z argumentów może być skalar. Takie wywołanie powoduje pomnożenie każdego elementu drugiego argumentu przez ten skalar. Przekazanie skalara do `np.matmul` spowoduje natomiast błąd.

Iloczyn Kroneckera

Ostatnią formą mnożenia macierzy, którą omówię w tej książce, będzie **iloczyn Kroneckera**. W trakcie obliczania iloczynu macierzowego mieszamy ze sobą poszczególne elementy macierzy, mnożąc je przez siebie. W przypadku iloczynu Kroneckera elementy jednej macierzy mnożymy przez całą drugą macierz, co powoduje, że wyjściowa macierz jest większa od obu macierzy wejściowych. Przy okazji omawiania iloczynu Kroneckera warto wspomnieć o **macierzach blokowych**, czyli macierzach składających się z mniejszych macierzy nazywanych blokami.

Kup ksi k Pole ksi k

Na przykład dla trzech poniższych macierzy

$$\boldsymbol{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \boldsymbol{B} = \begin{bmatrix} 11 & 22 \\ 33 & 44 \\ 55 & 66 \end{bmatrix} \boldsymbol{C} = \begin{bmatrix} 111 \\ 222 \\ 333 \end{bmatrix} \tag{5.10}$$

w następujący sposób możemy zdefiniować macierz blokową $\boldsymbol{M}$:

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{A} & \boldsymbol{B} & \boldsymbol{C} \\ \boldsymbol{B} & \boldsymbol{C} & \boldsymbol{A} \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 11 & 22 & 111 \\ 4 & 5 & 6 & 33 & 44 & 222 \\ 7 & 8 & 9 & 55 & 66 & 333 \\ 11 & 22 & 111 & 1 & 2 & 3 \\ 33 & 44 & 222 & 4 & 5 & 6 \\ 55 & 66 & 333 & 7 & 8 & 9 \end{bmatrix}$$

Każdy element $\boldsymbol{M}$ jest mniejszą macierzą. Elementy $\boldsymbol{M}$ są ułożone jedne obok drugich.

Iloczyn Kroneckera najłatwiej zdefiniować za pomocą wizualizacji wykorzystującej macierze blokowe. Iloczyn Kroneckera macierzy $\boldsymbol{A}$ i $\boldsymbol{B}$ oznacza się zazwyczaj symbolem $\boldsymbol{A} \otimes \boldsymbol{B}$. Iloczyn ten to

$$\boldsymbol{A} \otimes \boldsymbol{B} = \begin{bmatrix} a_{00}\boldsymbol{B} & a_{01}\boldsymbol{B} & \cdots & a_{0,n-1}\boldsymbol{B} \\ a_{10}\boldsymbol{B} & a_{11}\boldsymbol{B} & \cdots & a_{1,n-1}\boldsymbol{B} \\ \vdots & \vdots & & \vdots \\ a_{m-1,0}\boldsymbol{B} & a_{m-1,1}\boldsymbol{B} & \cdots & a_{m-1,n-1}\boldsymbol{B} \end{bmatrix}$$

gdzie $\boldsymbol{A}$ jest macierzą o wymiarach $m \times m$. Wynikowa macierz jest macierzą blokową ze względu na obecność $\boldsymbol{B}$, a zatem po całkowitym rozpisaniu powyższej macierzy iloczyn Kroneckera da w wyniku macierz większą od $\boldsymbol{A}$ i $\boldsymbol{B}$. Uwaga, w odróżnieniu od mnożenia macierzy iloczyn Kroneckera jest zdefiniowany dla macierzy o dowolnych rozmiarach. Na przykład iloczyn Kroneckera macierzy $\boldsymbol{A}$ i $\boldsymbol{B}$ z równania 5.10 to:

$$\boldsymbol{A} \otimes \boldsymbol{B} = \begin{bmatrix} (1)\boldsymbol{B} & (2)\boldsymbol{B} & (3)\boldsymbol{B} \\ (4)\boldsymbol{B} & (5)\boldsymbol{B} & (6)\boldsymbol{B} \\ (7)\boldsymbol{B} & (8)\boldsymbol{B} & (9)\boldsymbol{B} \end{bmatrix} = \begin{bmatrix} 11 & 22 & 22 & 44 & 33 & 66 \\ 33 & 44 & 66 & 88 & 99 & 132 \\ 55 & 66 & 110 & 132 & 165 & 198 \\ 44 & 88 & 55 & 110 & 66 & 132 \\ 132 & 176 & 165 & 220 & 198 & 264 \\ 220 & 264 & 275 & 330 & 330 & 396 \\ 77 & 154 & 88 & 176 & 99 & 198 \\ 231 & 308 & 264 & 352 & 297 & 396 \\ 385 & 462 & 440 & 528 & 495 & 594 \end{bmatrix}$$

Zauważ, że iloczyn Kroneckera oznaczyłem symbolem $\otimes$. To pewna konwencja, choć symbol $\otimes$ jest czasami nadużywany i stosuje się go także do innych celów. Ja na przykład wykorzystałem go w tym rozdziale do oznaczenia iloczynu zewnętrznego dwóch wektorów. W NumPy do wyznaczenia iloczynu Kroneckera możesz wykorzystać funkcję `np.kron`.

Kup ksi k Pole ksi k

Podsumowanie

W tym rozdziale przedstawiłem skalary, wektory, macierze i tensory, czyli obiekty matematyczne wykorzystywane w uczeniu głębokim. Następnie wyjaśniłem Ci arytmetykę tensorów, a w szczególności operacje na wektorach i macierzach. Pokazałem Ci, jak wykonuje się operacje na tych obiektach zarówno na papierze, jak i za pomocą NumPy.

Rozdział ten nie kończy jednak mojego omówienia algebry liniowej. W następnym rozdziale zagłębimy się w macierze i ich własności. Przedstawię w nim kilka rzeczy, które warto wiedzieć o macierzach, i pokażę Ci, co możemy z nimi zrobić.

Kup książkę Poleć książkę

Skorowidz

A

algebra liniowa, 127, 153, 332
algorytm
  metody gradientu prostego, 270, 297, 307, 324
  propagacji wstecznej, 270
algorytmy optymalizacji, 237
analiza głównych składowych, PCA, 179

B

bezwładność, 312
biblioteka
  Pillow, 248
  scikit-learn, 36
błąd, 281
  standardowy, 100
brakujące dane, missing data, 106
broadcasting, 134

C

centralne twierdzenie graniczne, 77
centroid, 174
CNN, convolutional neural network, 253

D

dane
  antagonistyczne, 176
  ilorazowe, 91
  interwałowe, 90
  nominalne, 90
  porządkowe, 90
  wejściowe spoza dziedziny, 176
dopełnienie algebraiczne, 162
dostrajanie hiperparametrów, 310
dylemat Monty'ego Halla, 42
dywergencja Kullbacka-Leiblera, 176

E

ekstremum funkcji, 202, 205
entropia względna, 176

F

funkcja
  gęstości prawdopodobieństwa, 75
  masy prawdopodobieństwa, 69
  np.ones, 29
  np.zeros, 29
  ReLU, 242
  sigmoidalna, 253, 271
  straty
    znajdowanie minimum, 237
funkcje
  aktywacji, 242, 271
  hesjany, 228, 235, 237
  macierzowe
    przyjmujące skalar, 221
  maksima, 202, 236
  minima, 202, 236
  przestępne, 238
  skalarne
    przyjmujące macierz, 222, 227
    z argumentem wektorowym, 220, 223
  wektorowe
    przyjmujące wektor, 221, 226
    z argumentem skalarnym, 219, 225
  znajdowanie pierwiastków
    metoda Newtona, 231

Kup ksi k

Pole ksi k

G

gradient, 210, 220
obliczanie, 210
wizualizacja, 212
graf obliczeniowy, 292
symbol-na-liczbę, 293
symbol-na-symbol, 293
granica, 192

H

hesjan, 228, 235, 237
hiperparametr, 309
hipoteza
alternatywna, 117
zerowa, 117
hipotezy
testowanie, 115
histogram, 64, 67

I

iloczyn
Hadamarda, 134, 283
kartezjański, 143
Kroneckera, 149
skalarny, 138
wektorowy, 143
wewnętrzny, 138
zewnętrzny, 142
instalowanie Pythona, 24–26
istotność statystyczna, 119

J

jądro konwolucji, 254

K

klasyfikator najbliższego centroidu, 174
konwolucja, 254
poprawna, 256
w dwóch wymiarach, 257
w jednym wymiarze, 254
korelacja, 109
krzyżowa, 256
Spearmana, 113
krok, stride, 258
krzywa Gaussa, 75
kwantyl, 101

L

L-normy, 170

M

macierz, 129
blokowa, 149
błędu, 280
diagonalna, 159
gradientu, 223
Hessego, 228, 235
Jacobiego, 228
jednostkowa, 158
jedynek, 157
kowariancji, 171
kwadratowa, 147, 154
obliczanie wyznacznika, 160
nieosobliwa, 164
odwracalna, 164
odwrotna, 164
ortogonalna, 165
osobliwa/singularna, 164
permutacji, 163
rotacji/obrotów, 154
symetryczna, 165
określoność, 166
trójkątna, 159
unitarna, 165
wag, 281
zerowa, 157
macierze
iloczyn Kroneckera, 149
mnożenie, 144
mnożenie w NumPy, 147
notacja macierzowa, 147
potęgowanie, 157
rozkład SVD, 183
transpozycja, 155
wartości własne, 166
wyznaczanie śladu, 156
Matplotlib, 34
MCC, 317
mediana, 95
bezwzględnych odchyleń, MAD, 99

Kup książkę
Poleć książkę

metoda
Adadelta, 326
Adagrad, 325
gradientu prostego, 270, 297
adaptacyjna, 324
jednowymiarowa, 298
stochastyczna, 308
w dwóch wymiarach, 302
wizualizacja, 300, 304, 305
z pędem, 313, 315
z pojedynczym minimum, 302
z wieloma minimami, 306
Newtona, 231
RMSprop, 324, 325
metody
drugiego rzędu, 237
optymalizacji, 327
pierwszego rzędu, 237
miary
odległości, 170
zmienności, 97
minipartie, minibatches, 248

N

nachylenie prostej, 190
naiwny klasyfikator Bayesa, 84
norma wektorowa, 170
notacja dużego O, 240
NumPy, 27
definiowanie tablic, 27
mnożenie macierzy, 147
tablice
indeksowanie, 30
odczyt i zapis, 32

O

ocena wyników testu, 121
odchylenie standardowe, 98, 100
odległość
Czebyszewa, 171
euklidesowa, 171
Mahalanobisa, 174
odwrotna dystrybuanta, 122
opadanie gradientowe, 298
operacje tablicowe, 133
optymalizacja, 237, 327
osadzenie, embeddings, 142

P

pakiet
Matplotlib, 34
NumPy, 27
SciPy, 33
paradoks dnia urodzin, 49
percentyl, 101
perceptron wielowarstwowy, 37
pęd, 311
Adam, 326
Niestierowa, 321
w dwóch wymiarach, 314
w jednym wymiarze, 312
Pillow, 248
pochodna, 191, *Patrz także* reguły różniczkowania
funkcji
aktywacji, 242
macierzowej, 221
potęgowej, 194
sigmoidalnej, 273
skalarnej, 220–223, 227
stałej, 193
trygonometrycznych, 197
wektorowej, 219, 225–228
wykładniczej, 199
iloczynu, 195
ilorazu, 195
kierunkowa, 212
logarytmu, 199
złożenia, 196, 198
pochodne
cząstkowe, 206, 272
mieszane, 207
reguła łańcuchowa, 208
wag, 283
wyrazów wolnych, 283
operacji elementarnych, 241
w macierzowym rachunku różniczkowym, 218
pole
skalarne, 210, 220
wektorowe, 210
prawdopodobieństwo, 40, 331
a posteriori, 82
a priori, 82, 83
brzegowe, 55
całkowite, 54

Kup książkę Poleć książkę

prawdopodobieństwo
łączne, 55
tabele, 56
reguła
dodawania, 47, 48
łańcuchowa, 60
mnożenia, 47
rozkład
ciągły, 73
dwumianowy, 68
dyskretny, 64
jednostajny, 64
Poissona, 70
zero-jedynkowy, 70
warunkowe, 52
zdarzenia, 44
prawo wielkich liczb, 80
propagacja wsteczna, 269
błędu, 281
w sieci w pełni połączonej, 280
prosta
równanie, 190
współczynnik nachylenia, 190
próbka, 40
przedziały ufności, 120
przekształcenia afiniczne, 154
przestrzeń próbek, 40
pseudoodwrotność, 183
Moore'a-Penrose'a, 186
punkt
przegięcia, 202
siodłowy, 202
stacjonarny, 191, 202, 203
Python
implementacja sieci, 285
instalowanie, 24–26
propagacja wsteczna, 274
testowanie modelu, 278
typy danych, 28
uczenie modelu, 278

R

rachunek różniczkowy, 189, 332
macierzowy, 217
reguła
Bayesa, 81
dodawania, 47, 48
łańcuchowa, 195
dla pochodnych cząstkowych, 208
dla prawdopodobieństwa, 60
mnożenia, 47
prawej ręki, 143
reguły różniczkowania, 201
ReLU, recitfied linear unit, 242
rozkład
dwumianowy, 68
Poissona, 70
prawdopodobieństwa, 63
według wartości osobliwych, SVD, 183
zero-jedynkowy, 70
rozstęp międzykwartylowy, IQR, 105
rozwinięcie
Laplace'a, 162
w szereg Taylora, 238
równania różniczkowe autonomiczne, 229

S

scikit-learn, 36
SciPy, 33
SGD, stochastic gradient descent, 309
sieci neuronowe
głębokie, 247
konwolucyjne, 247
przepływ danych, 253, 264
warstwy konwolucyjne, 259
warstwy łączące, 262
warstwy w pełni połączone, 263
tradycyjne, 246
przepływ danych, 249
sieczna, 191
silnik ewaluacji grafów, 293
skalar, 128
splot, 254
sprzężenie hermitowskie, 165
statystyka, 89, 331
statystyki podsumowujące, 92
stochastyczna metoda gradientu prostego, SGD, 309
stopnie swobody, 119
styczna, 191
SVD, singular value decomposition, 183
zastosowania, 185
szereg Taylora, 238
sztuczka z wyrazem wolnym, bias trick, 155

Ś

średnia
 arytmetyczna, 93
 geometryczna, 93
 harmoniczna, 94
średnie odchylenie bezwzględne, 97

T

tabele krzyżowe, 56
tablice dwuwymiarowe, 29
tensor, 130, 133
TensorFlow, 293
test
 dwustronny, 118
 t, 118
 t Welcha, 119
 U Manna-Whitneya, 123
testowanie
 hipotez, 115
 modelu, 278
twierdzenie Bayesa, 80, 84
tworzenie
 sieci neuronowej, 36, 271
 wykresów, 34
typy danych, 28

U

uczenie
 głębokie, 333
  norma i odległość, 170
  pochodne operacji elementarnych, 241
  zastosowanie SVD, 185
 maszynowe, 170
  dane nominalne, 92
  twierdzenie Bayesa, 84
 modelu, 251, 277, 278
  metody z pędem, 315
 online, 309
 wsadowe, batch training, 289
układ
 licznikowy, numerator layout, 218
 mianownikowy, denominator layout, 218

W

wariancja z próby
 nieobciążona, 98
 obciążona, 97
warstwy
 konwolucyjne, 247, 259
 łączące, pooling layer, 262
wartości
 osobliwe, 184
 własne macierzy, 167
wartość
 krytyczna, 120
 oczekiwana, 110
wąsy, whiskers, 105
wektor, 128
 jednostkowy, 136
 styczny, 220
 wierszowy, 147
wektory, 166
 długość, 136
 iloczyn skalarny, 138
 iloczyn wektorowy, 143
 iloczyn zewnętrzny, 142
 rzutowanie, 140
 transpozycja, 137
 własne, 167
wielkość efektu, 121
wielomian charakterystyczny, 168
współczynnik
 korelacji, 113
  Matthewsa, MCC, 94, 317
  Pearsona, 109
  Spearmana, 113
 nachylenia prostej, 192
 rozpadu, 324
wykres
 3D, 36
 pudełkowy, 100, 104, 108

Z

zakres wartości, 97
zbiór danych
 FMNIST, 317
 iris, 275
 MNIST, 290
zdarzenia, 40
 niezależne, 47
 przeciwne, 44
 rozłączne, 47
zmienna losowa
 ciągła, 41
 dyskretna, 41

Kup ksi k Pole ksi k

PROGRAM PARTNERSKI

GRUPY HELION

1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!
http://program-partnerski.helion.pl

CHCESZ ZROZUMIEĆ SIECI NEURONOWE? ODPOWIEDZI SZUKAJ W MATEMATYCE!

Uczenie maszynowe niesie ze sobą obietnicę niezwykłych wynalazków: od samochodów autonomicznych po systemy medyczne diagnozujące choroby lepiej niż doświadczeni lekarze, ale także daje pole do rozwijania dziesiątków innych mniej lub bardziej niepokojących innowacji. Dziś do budowania systemów uczenia maszynowego można użyć wygodnych frameworków, jednak rzeczywiste zrozumienie uczenia głębokiego wymaga znajomości kilku koncepcji matematycznych.

Koncepcje te zostały przystępnie wyjaśnione właśnie w tej książce. W szczególności zapoznasz się z praktycznymi aspektami probabilistyki, statystyki, algebry liniowej i rachunku różniczkowego. Prezentacji tych zagadnień towarzyszą fragmenty kodu w Pythonie i praktyczne przykłady zastosowań w uczeniu głębokim. Rozpoczniesz od zapoznania się z podstawami, takimi jak twierdzenie Bayesa, a następnie przejdziesz do bardziej zaawansowanych zagadnień, w tym uczenia sieci neuronowych przy użyciu wektorów, macierzy i pochodnych. Dwa ostatnie rozdziały dadzą Ci szansę użycia nowej wiedzy do zaimplementowania propagacji wstecznej i metody gradientu prostego — dwóch podstawowych algorytmów napędzających rozwój sztucznej inteligencji.

W książce między innymi:

- zapewnienie większej wydajności działania aplikacji
- automatyczne skalowanie i mechanizm równoważenia obciążenia
- zastosowanie statystyki do zrozumienia danych i oceny modeli
- prawidłowe korzystanie z reguł prawdopodobieństwa
- użycie wektorów i macierzy do przesyłania danych w sieciach neuronowych
- algebra liniowa w analizie głównych składowych i rozkładu według wartości osobliwych
- gradientowe metody optymalizacji, takie jak RMSprop, Adagrad i Adadelta

Dr Ronald T. Kneusel zawodowo zajmuje się uczeniem maszynowym od 2003 roku. W 2016 roku obronił doktorat z tej dziedziny na Uniwersytecie Kolorado w Boulder. Jest autorem kilku książek, w tym *Deep learning. Praktyczne wprowadzenie z zastosowaniem środowiska Pythona*.

Helion

helion.pl

HELION S.A.
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

KOD KORZYŚCI
Sięgnij po więcej! ▶

ISBN 978-83-289-1016-4

9 788328 910164

Cena: 89,00 zł