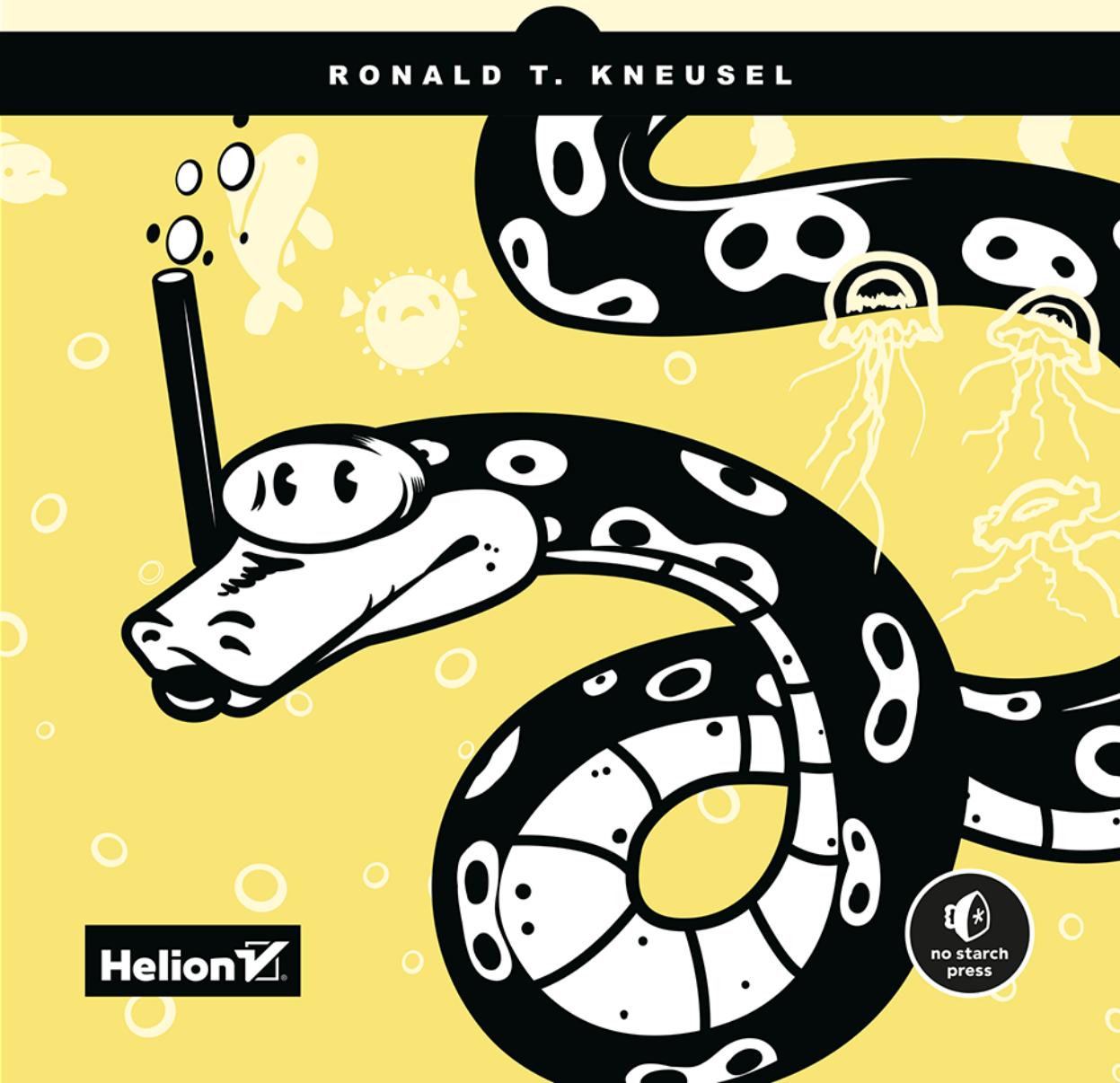
DEEP LEARNING

PRAKTYCZNE WPROWADZENIE
Z ZASTOSOWANIEM ŚRODOWISKA PYTHONA

RONALD T. KNEUSEL



Helion

no starch press

Tytuł oryginału: Practical Deep Learning A Python-Based Introduction

Tłumaczenie: Krzysztof Sawka

ISBN: 978-83-283-8859-8

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

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

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Printed in Poland.

Spis treści

1

Pierwsze kroki

Rozdział ten stanowi wprowadzenie do naszego środowiska operacyjnego oraz zawiera szczegóły jego konfiguracji. Znajdziesz tu także powtórzenie pewnych elementów matematyki, z których będziemy korzystać w dalszej części książki. Ostatni podrozdział został poświęcony krótkiemu omówieniu procesorów graficznych (ang. graphic processing unit — GPU), które mają duże znaczenie w uczeniu głębokim. Na szczęście w naszym przypadku nie odgrywają istotnej roli, dlatego nie musisz się martwić; korzystanie z tej książki nie będzie wiązało się z niespodziewanymi olbrzymimi kosztami.

Środowisko operacyjne

W tym podrozdziale zajmiemy się opisem środowiska, z którego będziemy korzystać aż do końca książki. Zakładamy tu, że używasz 64-bitowego Linuksa. Rodzaj dystrybucji nie jest zbyt istotny, ale żeby dodatkowo ułatwić wybór, przyjmijmy także, że korzystamy z Linuksa Ubuntu 20.04. Mając na uwadze znakomite wsparcie dystrybucji Ubuntu, uważamy, że nowsza wersja nie powinna powodować problemów. Python jest naszym *lingua franca*, powszechnym językiem uczenia maszynowego. Dokładniej mówiąc, będziemy używać Pythona w wersji 3.8.2; jest to wersja dostępna w Ubuntu 20.04.

Przyjrzyjmy się pokrótce używanym przez nas narzędziom Pythona.

NumPy

NumPy jest biblioteką Pythona dodającą do środowiska przetwarzanie tablicowe. Listy Pythona można wykorzystywać jako jednowymiarowe tablice, jednak w praktyce są zbyt powolne i ograniczone. Biblioteka NumPy dodaje funkcje tablicowe, których brakuje Pythonowi; są one niezbędne w wielu zastosowaniach naukowych. NumPy stanowi bibliotekę bazową wykorzystywaną przez wszystkie pozostałe biblioteki, z których będziemy korzystać.

scikit-learn

Wszystkie omówione w tej książce tradycyjne modele uczenia maszynowego dostępne są w znakomitej bibliotece **scikit-learn** (czyli `sklearn`, jak jest nazywana po wczytaniu do Pythona). Zwróć uwagę, że zapis nazwy pozbawiony jest dużych liter, gdyż takiej właśnie notacji używają konsekwentnie sami twórcy w dokumentacji biblioteki. Biblioteka ta wykorzystuje tablice NumPy. Implementuje ona interfejs ustandaryzowany dla wielu różnych modeli uczenia maszynowego, a także całe mnóstwo funkcji, o których nie zdążę nawet wspomnieć. Bardzo polecam zapoznanie się z oficjalną dokumentacją sklearn (*https://scikit-learn.org/stable/index.html*) w miarę ustawicznego zapoznawania się z uczeniem maszynowym i narzędziami służącymi do projektowania modeli.

Keras i TensorFlow

Samo w sobie zrozumienie uczenia głębokiego nie jest łatwe, nie wspominając nawet o jego wydajnej oraz poprawnej implementacji, dlatego zamiast próbować rozpisywać od podstaw splotową sieć neuronową, skorzystamy z popularnego, aktywnie rozwijanego narzędzia. Od samego początku społeczność uczenia głębokiego wspiera rozwój narzędzi ułatwiających korzystanie z sieci głębokich i udostępnia otwarte narzędzia objęte bardzo przyjaznymi licencjami. W czasie pisania tej książki dostępnych jest wiele popularnych narzędzi przygotowanych z myślą o Pythonie. Są to między innymi:

- Keras,
- PyTorch,
- Caffe,
- Caffe2,
- Apache MXnet.

Niektóre z tych narzędzi są prężnie rozwijane, o innych stopniowo zapominamy. Obecnie jednak jednym z mających najbardziej oddaną społeczność jest biblioteka Keras, wykorzystująca z kolei bibliotekę TensorFlow, dlatego będziemy z niej korzystać w tej książce.

Keras (*https://keras.io/*) jest biblioteką uczenia głębokiego wykorzystującą do działania bibliotekę **TensorFlow** (*https://www.tensorflow.org/*). Z kolei TensorFlow to otwarty produkt firmy Google, implementujący kluczowe funkcje głębokich sieci neuronowych na wielu różnych platformach. Wybraliśmy bibliotekę

Keras nie tylko z powodu popularności i błyskawicznego rozwoju, lecz także dlatego, że jest prosta w obsłudze. Naszym celem jest zrozumienie uczenia głębokiego w stopniu umożliwiającym implementowanie modeli, a następnie korzystanie z nich przy minimum złożoności programistycznej.

Instalacja narzędzi

Nie jesteśmy w stanie opisać procesu instalacji narzędzi na wszystkich systemach operacyjnych i na wszystkich konfiguracjach sprzętowych. Zamiast tego zaprezentujemy instrukcje opisujące krok po kroku proces instalacji dla określonego systemu operacyjnego będącego naszym systemem referencyjnym. Etapy te, wraz z minimalnymi wymaganymi wersjami bibliotek, powinny wystarczyć większości czytelników do przygotowania działającego środowiska roboczego.

Pamiętaj, że zakładamy pracę w środowisku linuksowym, dokładniej w dystrybucji Ubuntu 20.04. Ubuntu jest szeroko rozpowszechnioną dystrybucją Linuksa i działa na niemal każdym nowym komputerze. Inne dystrybucje Linuksa również sprawdzą się jako środowisko pracy, podobnie jak system macOS, ale podane tu instrukcje są przeznaczone dla dystrybucji Ubuntu. Społeczność uczenia maszynowego raczej nie korzysta z systemu Windows. Istnieją jednak narzędzia przystosowane do tego systemu; zatem odważny czytelnik może spróbować także tam swych sił[1].

Świeżo zainstalowana dystrybucja Ubuntu 20.04 zawiera domyślnie środowisko Python 3.8.2. W celu zainstalowania pozostałych pakietów musimy wejść do powłoki i wpisać poniższe polecenia w podanej kolejności:

```
$ sudo apt-get update
$ sudo apt-get install python3-pip
$ sudo apt-get install build-essential python3-dev
$ sudo apt-get install python3-setuptools python3-numpy
$ sudo apt-get install python3-scipy libatlas-base-dev
$ sudo apt-get install python3-matplotlib
$ pip3 install scikit-learn
$ pip3 install tensorflow
$ pip3 install pillow
$ pip3 install h5py
$ pip3 install keras
```

Po zakończeniu instalacji będziemy dysponować następującymi wersjami narzędzi i bibliotek:

[1] Nie jest to do końca prawda. System Windows jest dobrze wspierany pod względem środowiska Python i bibliotek uczenia maszynowego. Osoby zainteresowane mogą chcieć zainstalować środowisko **Anaconda**, a następnie dołączyć wymagane biblioteki; opisane w książce przykłady działają zarówno w Linuksie, jak i w systemie Windows — *przyp. tłum.*

```
NumPy 1.17.4
sklearn 0.23.2
keras 2.4.3
tensorflow 2.2.0
pillow 7.0.0
h5py 2.10.0
matplotlib 3.1.2
```

Biblioteka `pillow` służy do przetwarzania obrazów, `h5py` pozwala na pracę z plikami w formacie HDF5, natomiast `matplotlib` umożliwia tworzenie wykresów. **HDF5** to ogólny, hierarchiczny format plików służący do przechowywania danych naukowych. Biblioteka Keras przechowuje w tym formacie parametry modelu.

W dwóch następnych podrozdziałach znajdziesz łagodne wprowadzenie do pewnych pojęć matematycznych spotykanych w dalszej części książki.

Podstawy algebry liniowej

Przyjrzymy się teraz wektorom i macierzom. Opisujący te twory dział matematyki nazywany jest **algebrą liniową** lub teorią macierzy. Jak można się domyślić, algebra liniowa jest całkiem skomplikowaną dziedziną. Nam wystarczy wiedzieć, czym jest wektor oraz macierz, a także jak mnożymy dwa wektory, dwie macierze lub wektor przez macierz. Przekonamy się później, że dzięki temu uzyskujemy bardzo przydatny mechanizm implementowania określonych modeli, zwłaszcza sieci neuronowych.

Zacznijmy od wektorów.

Wektory

Wektor (ang. *vector*) jest jednowymiarową listą wartości liczbowych. W ujęciu matematycznym może przybierać on następującą postać:

$$a = [0,1,2,3,4]$$

Trzeci element jest zapisywany jako $a_2 = 2$. Zwróć uwagę, że korzystamy z konwencji programistycznej, zgodnie z którą indeksy wyznaczamy od wartości 0, zatem a_2 daje nam trzeci element wektora.

Powyższy wektor zapisaliśmy poziomo, dlatego nazywamy go **wektorem wierszowym** (ang. *row vector*). Jednak wektory stosowane w wyrażeniach matematycznych zazwyczaj są zapisywane w pionie:

$$a = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

Wektor zapisywany w pionie nazywany jest **wektorem kolumnowym** (ang. *column vector*). W naszym przykładzie zawiera on pięć elementów, dlatego mówimy, że jest pięcioelementowym wektorem kolumnowym. W tej książce wektor symbolizuje pojedynczą **próbę** (ang. *sample*), czyli jeden zestaw cech wprowadzanych na wejściu modelu.

W ujęciu matematycznym wektory służą do reprezentowania punktów w przestrzeni. Jeżeli weźmiemy dwuwymiarową płaszczyznę kartezjańską, to wyznaczamy położenie danego punktu za pomocą dwuelementowego wektora *(x, y)*, gdzie *x* stanowi odległość wzdłuż osi *x*, natomiast *y* jest odległością wzdłuż osi *y*. Wektor taki określa punkt w dwóch wymiarach, mimo że sam jest jednowymiarowy. Gdybyśmy mieli punkt w trzech wymiarach, musielibyśmy użyć wektora trójelementowego *(x, y, z)*.

W uczeniu maszynowym wektory służą często do symbolizowania sygnałów wejściowych do modelu, dlatego pracujemy z dziesiątkami, a nawet setkami wymiarów. Oczywiście nie jesteśmy w stanie zaznaczać ich na wykresie jako punktów w przestrzeni, ale tym właśnie są w ujęciu matematycznym. Jak się przekonamy, niektóre modele, na przykład algorytm *k* najbliższych sąsiadów, wykorzystują wektory cech właśnie jako punkty w wielowymiarowej przestrzeni.

Macierze

Macierz (ang. *matrix*) jest dwuwymiarową tablicą zawierającą wartości liczbowe, w której dany element określamy za pomocą numeru wiersza i kolumny. Poniżej widzimy przykładową macierz:

$$a = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

Jeżeli odnosimy się do liczby 6, to zapisujemy $a_{1,2} = 6$. Ponownie indeksujemy tu od wartości 0. Macierz *a* zawiera po trzy wiersze i kolumny, dlatego nazywamy ją macierzą 3×3.

Mnożenie wektorów i macierzy

Mnożenie dwóch wektorów jest niezmiernie proste; wystarczy pomnożyć elementy znajdujące się na tych samych pozycjach. Na przykład:

$$[1,2,3] \cdot [4,5,6] = [4,10,18]$$

Jest to najczęściej spotykany sposób mnożenia tablicy podczas używania biblioteki NumPy i będziemy z niego bardzo często korzystać w kolejnych rozdziałach. Jednak w typowej matematyce nie jest on zbyt często używany.

W przypadku klasycznie matematycznego iloczynu wektorów musimy brać pod uwagę fakt, czy są one wierszowe, czy kolumnowe. Będziemy pracować na dwóch wektorach: $A = (a, b, c)$ i $B = (d, e, f)$, które, zgodnie z przyjętą matematyczną konwencją, są domyślnie kolumnowe. Dodanie indeksu górnego *T*

przekształca wektor kolumnowy w wierszowy. Matematycznie dopuszczalnymi sposobami mnożenia A i B są:

$$AB^T = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \begin{bmatrix} d & e & f \end{bmatrix} = \begin{bmatrix} ad & ae & af \\ bd & be & bf \\ cd & ce & cf \end{bmatrix}$$

Operacja ta nazywana jest **iloczynem zewnętrznym** (ang. *outer product*). Z kolei operacja

$$A^T B = \begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} d \\ e \\ f \end{bmatrix} = ad + be + cf$$

jest **iloczynem wewnętrznym** (ang. *inner product*), znanym także jako **iloczyn skalarny** (ang. *dot product*). Zwróć uwagę, że w wyniku iloczynu zewnętrznego otrzymujemy macierz, a rezultatem iloczynu wewnętrznego jest pojedyncza liczba (**skalar**).

Podczas mnożenia macierzy i wektora ten drugi zazwyczaj znajduje się po prawej stronie macierzy. Operację można przeprowadzić tylko wtedy, gdy liczba kolumn macierzy jest równa liczbie elementów wektora (który również w tym przypadku powinien być kolumnowy). W konsekwencji otrzymujemy wektor o liczbie elementów równej liczbie wierszy macierzy ($ax + by + cz$ jest jednym elementem).

Na przykład:

$$\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} ax + by + cz \\ dx + ey + fz \end{bmatrix}$$

Pomnożyliśmy tu macierz 2×3 przez wektor kolumnowy 3×1, dzięki czemu otrzymaliśmy wektor 2×1. Zwróć uwagę, że liczba kolumn macierzy i wierszy wektora musi być taka sama, w przeciwnym razie nie można wykonać operacji mnożenia. Ponadto wartości w wektorze wynikowym są sumami iloczynów macierzy i wektora. Ta sama reguła dotyczy iloczynu dwóch macierzy:

$$\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix} \begin{bmatrix} A & B \\ C & D \\ E & F \end{bmatrix} = \begin{bmatrix} aA + bC + cE & aB + bD + cF \\ aA + eC + fE & dB + eD + fF \end{bmatrix}$$

Iloczyn macierzy 2×3 przez macierz 3×2 dał nam macierz 2×2.

Podczas omawiania sieci splotowych będziemy pracować z tablicami trój-, a nawet czterowymiarowymi. Obiekty takie są nazywane **tensorami** (ang. *tensor*). Jeśli wyobrazisz sobie stos macierzy mających takie same rozmiary, otrzymasz tensor trójwymiarowy; w takim przypadku pierwszy indeks służy do określenia interesującej nas macierzy, a za pomocą dwóch pozostałych wyznaczamy interesujący nas element tejże macierzy. Z kolei jeżeli ułożysz stos trójwymiarowych tensorów,

to otrzymasz tensor czterowymiarowy, w którym pierwszy indeks wyznacza interesujący nas tensor trójwymiarowy.

Główne wnioski z tego podrozdziału są takie, że wektory mają jeden wymiar, macierze mają dwa wymiary, istnieją pewne reguły mnożenia tych obiektów, a także że nasze biblioteki będą korzystać w pewnym momencie z czterowymiarowych tensorów. Jeszcze wrócimy do tych wniosków w dalszej części książki.

Statystyka i prawdopodobieństwo

Działy statystyki i probabilistyki są tak rozległe, że często można jedynie wspomnieć o nich zaledwie kilka słów albo od razu napisać poświęconą im całą książkę czy dwie. Z tego powodu wspomnę tu jedynie o niektórych kluczowych aspektach i pozostawię Ci dokształcenie się w tych tematach zależnie od Twoich potrzeb. Zakładam, że na podstawie rzutów monetą i kostką pozyskałeś pewną elementarną wiedzę na temat prawdopodobieństwa.

Statystyka opisowa

Podczas przeprowadzania eksperymentów musimy zgłaszać wyniki w jakiś znaczący sposób. W naszym przypadku oznacza to zazwyczaj określanie wartości średniej plus minus wartość znana jako **błąd standardowy średniej** (ang. *standard error of the mean* — *SE*). Zdefiniujmy ten błąd na przykładzie.

W przypadku dużej liczby pomiarów x, na przykład długości określonej części kwiatu, możemy obliczyć średnią ($\overline{x}$) przez dodanie wszystkich wartości i podzielenie tej sumy przez liczbę wszystkich dodanych wartości. Po otrzymaniu średniej możemy obliczyć średnią rozpiętość poszczególnych wartości wokół średniej poprzez odjęcie każdej wartości od średniej, podniesienie wyniku do kwadratu, a następnie zsumowanie tak otrzymanych wartości i podzielenie otrzymanej sumy przez liczbę elementów minus jeden. W ten sposób uzyskujemy **wariancję** (ang. *variance*). Pierwiastek kwadratowy z wariancji nazywany jest **odchyleniem standardowym** (σ, ang. *standard deviation*), do którego powrócimy za chwilę. Za pomocą odchylenia standardowego możemy policzyć błąd standardowy jako $SE = \sigma/\sqrt{n}$, gdzie n jest liczbą wartości użytych do obliczenia średniej. Im mniejsza jest wartość SE, tym wartości są ściślej upakowane wokół średniej. Możemy interpretować tę wartość jako niepewność dotyczącą średniej. Zgodnie z nią przewidujemy, że rzeczywista wartość średnia (której nie znamy) mieści się w przedziale między $\bar{x} - SE$ a $\bar{x} + SE$.

Czasami zastępujemy średnią medianą. **Mediana** (ang. *median*) jest wartością środkową, poniżej której znajduje się dokładnie połowa prób. Aby określić medianę danego zestawu wartości, najpierw sortujemy je numerycznie, a następnie wyznaczamy wartość środkową. W przypadku nieparzystej liczby elementów próby wartość tę wyznacza środkowy element, natomiast jeżeli mamy parzystą liczbę elementów, medianą jest wartość średnia między dwoma środkowymi elementami. Mediana czasami okazuje się lepsza od średniej, jeżeli próby nie mają

dobrego, równomiernego rozrzutu wokół średniej. Klasycznym przykładem są dochody. Kilka najbogatszych osób tak przesuwa średnią do góry, że przestaje ona mieć większe znaczenie. Rozsądniejszym rozwiązaniem okazuje się mediana, która ukazuje jedną połowę osób zarabiających mniej i drugą zawierającą ludzi o wyższych dochodach.

W późniejszych rozdziałach będziemy zajmować się **statystyką opisową** (ang. *descriptive statistics*). Są to wartości obliczone dla zestawu danych, pomagające zrozumieć jego własności. Przed chwilą wymieniłem trzy takie wartości: średnią, medianę i odchylenie standardowe. Nauczymy się je interpretować i zaznaczać na wykresach w celu lepszego zrozumienia zestawów danych.

Rozkłady prawdopodobieństwa

W tej książce będziemy korzystać z pojęcia **rozkładu prawdopodobieństwa** (ang. *probability distribution*). Można traktować go jako swoistą wyrocznię, która, zapytana, przepowie jakąś liczbę lub zbiór liczb. Na przykład podczas uczenia modelu korzystamy z liczb lub zbiorów liczb, które mierzymy; liczby te biorą się z danego rozkładu prawdopodobieństwa. Jest to tak zwany **rozkład główny** (ang. *parent distribution*). Możemy traktować go jako generator danych dostarczanych do modelu; w bardziej filozoficznym ujęciu jest to idealny zestaw danych, którego przybliżenie stanowią nasze dane.

Istnieje wiele różnych rozkładów prawdopodobieństwa; niektóre z nich mają nawet własne nazwy. Dwoma najpowszechniejszymi są rozkład jednostajny i normalny. Niewątpliwie miałeś już do czynienia z rozkładem jednostajnym: ma on miejsce podczas rzucania kostką do gry. Skoro kostka ma sześć ścian, to wiemy, że prawdopodobieństwo wyrzucenia dowolnej wartości od 1 do 6 jest jednakowe. Jeżeli rzucimy kością sto razy i spiszemy otrzymywane wyniki, to zauważymy, że każdy numer został wyrzucony w przybliżeniu jednakową liczbę razy oraz że na dłuższą metę łatwo uwierzyć, że wartości te się wyrównają.

Rozkład jednostajny (ang. *uniform distribution*) jest wyrocznią, która z jednakowym prawdopodobieństwem może udzielić nam każdej z dopuszczalnych odpowiedzi. W ujęciu matematycznym zapisujemy go jako $U(a, b)$, gdzie litera U oznacza rozkład jednostajny (od ang. *uniform*), natomiast a i b wyznaczają zakres wartości losowanych z tego rozkładu. Dopuszczalne są liczby rzeczywiste, chyba że jawnie wymusimy losowanie wyłącznie liczb całkowitych. Symbolicznie zapisujemy $x \sim U(0, 1)$, co oznacza, że x jest wartością zwracaną przez wyrocznię losującą z jednakowym prawdopodobieństwem dane z zakresu liczb rzeczywistych w przedziale $(0,1)$. Nawiasy otwarte („(” i „)”) przy oznaczaniu przedziałów oznaczają przedział otwarty, czyli taki, w którym wartości graniczne są wykluczone z przedziału, natomiast nawiasy zamknięte („[” i „]”) wyznaczają przedział domknięty, uwzględniający w przedziale wartość graniczną oznaczoną takim nawiasem. Zatem $U[0, 1)$ zwraca wartości w przedziale od 0 do 1, gdzie wartość 0 jest uwzględniona w przedziale, natomiast wartość 1 zostaje pominięta.

Wykres **rozkładu normalnego** (ang. *normal distribution*), zwanego także **rozkładem Gaussa** (ang. *Gaussian distribution*) przybiera kształt dzwonowaty; w takim

przypadku jedna wartość ma największe prawdopodobieństwo wystąpienia, natomiast prawdopodobieństwo wystąpienia innych wartości maleje wraz z oddalaniem się od tego punktu. Najbardziej prawdopodobną wartością jest średnia $\overline{x}$, natomiast parametrem określającym szybkość redukowania prawdopodobieństwa do zera (które jednak zawsze pozostaje niezerowe) jest odchylenie standardowe σ (sigma). W naszym przypadku jeżeli chcemy losować próbę z rozkładu normalnego, zapiszemy $x \sim N(\bar{x}, \sigma)$, co oznacza, że x jest losowana z rozkładu normalnego o średniej $\overline{x}$ i odchyleniu standardowym σ.

Testy statystyczne

Kolejnym zagadnieniem, które będzie pojawiać się od czasu do czasu, jest koncepcja **testu statystycznego** (ang. *statistical test*), czyli pomiaru służącego do określania prawdziwości danej hipotezy. Zazwyczaj hipoteza odnosi się do zbiorów pomiarów; mianowicie hipoteza zakłada, że dwa zbiory pomiarów wywodzą się z tego samego rozkładu głównego. Jeżeli statystyka obliczona na podstawie testu wykracza poza pewien określony zakres, to odrzucamy hipotezę i stwierdzamy, że mamy dowód na pochodzenie obydwu zestawów danych z dwóch *różnych* rozkładów.

Tutaj będziemy zazwyczaj używać **testu t**, powszechnego testu statystycznego, w którym przyjmujemy rozkład normalny danych. Skoro zakładamy, że rozkład danych jest normalny, co może, ale nie musi być prawdą, to test ten nazywamy **testem parametrycznym** (ang. *parametric test*).

Czasami będziemy korzystać z **testu U Manna-Whitneya**, który przypomina test t, gdyż pomaga określić, czy dane dwie próby pochodzą z tego samego rozkładu głównego, nie zakładamy tu jednak odgórnie określonego rozkładu danych. Jest to przykład **testu nieparametrycznego** (ang. *nonparametric test*).

Bez względu na to, czy test jest parametryczny, czy nieparametryczny, zawsze otrzymujemy ostatecznie tak zwaną **wartość *p*** (ang. *p-value*). Stanowi ona prawdopodobieństwo wystąpienia obliczonej wartości testu w przypadku, gdyby hipoteza o pochodzeniu prób z tego samego rozkładu głównego była prawdziwa. Mała wartość p stanowi dowód na nieprawdziwość hipotezy.

Zazwyczaj granicą wartości p jest 0,05, co oznacza, że w 1 na 20 przypadków możemy zmierzyć daną wartość testu statystycznego (testu t lub U Manna-Whitneya), nawet gdyby próby pochodziły z tego samego rozkładu głównego. Ostatnie lata udowodniły jednak, że taki próg jest zbyt łagodny. Gdy wartość p jest bliska granicy 0,05, ale jej nie przekracza, to zaczynamy uznawać, że może istnieć jakiś dowód zaprzeczający hipotezie. Jeżeli wartość p wynosi, powiedzmy, 0,001 lub nawet mniej, to uzyskujemy mocny dowód na pochodzenie prób z dwóch różnych rozkładów. W takim przypadku mówimy, że różnica jest **statystycznie istotna/znacząca** (ang. *statistically significant*).

Procesory graficzne (GPU)

Jedną z technologii umożliwiających rozwój współczesnego uczenia głębokiego są potężne **procesory graficzne** (ang. *graphics processing unit — GPU*). Są to miniaturowe komputery stanowiące część karty graficznej. Pierwotnie zostały zaprojektowane z myślą o grach komputerowych, ale potężne możliwości równoległego przetwarzania operacji zostały dostosowane do skrajnych wymogów obliczeniowych modeli głębokich sieci neuronowych. Mnóstwo postępów z ostatnich lat nie byłoby możliwych bez niesamowitych możliwości obliczeniowych układów GPU, które są dostępne nawet na prostych komputerach stacjonarnych. Liderem w tworzeniu układów GPU przeznaczonych do uczenia głębokiego jest firma NVIDIA, a dzięki technologii **CUDA** (ang. *Compute Unified Device Architecture*) odegrała kluczową rolę w sukcesie uczenia głębokiego. Nie byłoby przesadą stwierdzenie, że bez układów GPU uczenie głębokie w ogóle nie miałoby miejsca, a z pewnością nie byłoby tak rozpowszechnione.

Skoro o tym mowa, nie oczekujemy, że w omawianych modelach muszą być wykorzystywane układy GPU. Będziemy używać na tyle małych zestawów danych i modeli, że do ich uczenia w rozsądnie krótkim czasie wystarczy zwyczajny procesor komputera. Wymusiliśmy zresztą ten wybór na etapie instalowania pakietów, ponieważ wybraliśmy wersję biblioteki TensorFlow działającą jedynie na zwykłym procesorze.

Jeżeli dysponujesz kartą graficzną wyposażoną w architekturę CUDA i chcesz skorzystać z niej podczas pracy nad przykładami opisującymi uczenie głębokie, to możesz to zrobić, jeśli jednak nie masz takiego układu graficznego, nie planuj jego zakupu wyłącznie w tym celu. Jeśli korzystasz z układu GPU, przed instalacją pakietów zainstaluj najpierw prawidłowo architekturę CUDA, a następnie bibliotekę TensorFlow obsługującą karty graficzne. Biblioteka scikit-learn korzysta wyłącznie ze zwykłego procesora.

Podsumowanie

W niniejszym rozdziale zapoznaliśmy się z naszym środowiskiem operacyjnym. Następnie omówiliśmy kluczowe biblioteki Pythona, z których będziemy korzystać w dalszej części książki, a także przygotowaliśmy instrukcję instalacji tych narzędzi przy założeniu, że będziesz używać dystrybucji Ubuntu 20.04. Jak już wspomnieliśmy, narzędzia te działają równie dobrze w innych dystrybucjach Linuksa oraz systemach operacyjnych. Przeszliśmy dalej do krótkiego omówienia aparatu matematycznego, który będzie nam później potrzebny, a rozdział zakończyliśmy opisem znaczenia układów GPU oraz wyjaśnieniem, dlaczego nie są wymagane podczas pracy na omawianych tu modelach.

Następny rozdział został poświęcony podstawom Pythona.

Skorowidz

C

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

W

Z


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
GRUPA
Helion

UCZENIE GŁĘBOKIE: PRZYSZEDŁ CZAS NA TWÓJ PIERWSZY MODEL!

Uczenie głębokie fascynuje wielu inżynierów i praktyków. Mimo że systemy oparte na uczeniu maszynowym stosuje się w rozlicznych branżach, wciąż są uważane za niepokojącą technologię. Istotnie, w wypadku na przykład sieci neuronowych nie wiemy, czego dokładnie uczy się model. Możemy tylko ocenić, czy dobrze realizuje swoje zadanie. Wydaje się, że w sposobie pracy algorytmów uczenia głębokiego tkwi magia. Właśnie dlatego dobrze jest zająć się faktami i dowiedzieć się, na czym w rzeczywistości polega uczenie maszynowe, a zwłaszcza uczenie głębokie.

Ta książka jest przystępnym przewodnikiem po uczeniu maszynowym. Aby zrozumieć zawartą w niej treść, wystarczy podstawowa umiejętność programowania i znajomość matematyki na poziomie szkoły średniej. Znalazło się tu omówienie podstawowych pojęć i wyjaśnienie mechanizmów rządzących uczeniem głębokim. Dzięki lekturze dowiesz się, czym się charakteryzuje dobry zbiór danych uczących, jak ocenić skuteczność modelu i jak korzystać z takich modeli jak najbliższych sąsiadów, lasy losowe czy maszyna wektorów nośnych. Sporo miejsca poświęcono również sieciom neuronowym, mechanizmom ich działania i technikom treningu. I chociaż nie znajdziesz tutaj gotowych receptur, to zdobędziesz wiedzę potrzebną, by od podstaw zaprojektować działający model uczenia głębokiego.

W książce między innymi:

- budowanie dobrego zestawu danych uczących
- praca z bibliotekami scikit-learn i Keras
- klasyczne modele uczenia maszynowego
- mechanizm działania i uczenia sieci neuronowych
- modele wykorzystujące splotowe sieci neuronowe
- przygotowanie od podstaw działającego modelu

Dr Ronald T. Kneusel uczeniem maszynowym zajmuje się zawodowo od 2003 roku, specjalizuje się w dziedzinie przetwarzania obrazów. Jego fascynacja komputerami jednak trwa znacznie dłużej, bo od 1981 roku. Obecnie pracuje w spółce L3Harris Technologies, gdzie buduje systemy uczenia głębokiego. Jest autorem popularnych książek na temat pisania kodu i uczenia głębokiego.

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

KOD KORZYŚCI
Sięgnij po więcej! ▶

ISBN 978-83-283-8859-8

Cena: 99,00 zł

no starch press