Einführung

Wenn Sie schon einmal versucht haben, etwas über neuronale Netze und Deep Learning zu erfahren, ist Ihnen vermutlich aufgefallen, wie viele verschiedene Quellen es dazu gibt – von Blogposts über MOOCs (*Massive Open Online Courses*, z. B. die von Coursera und Udacity angebotenen) unterschiedlicher Qualität bis hin zu Büchern. Zumindest hatte ich diesen Eindruck, als ich vor einigen Jahren begann, mich mit diesem Thema zu beschäftigen. Trotzdem ist es recht wahrscheinlich, dass Ihnen bei den bisher bekannten Erklärungen neuronaler Netze etwas fehlt. Mir ging es am Anfang genauso: Die verschiedenen Erklärungen wirkten, als versuchten Blinde, Teile eines Elefanten zu beschreiben (*https://oreil.ly/r5YxS*), ohne dass einer von ihnen den ganzen Elefanten beschreibt. Deshalb habe ich dieses Buch geschrieben.

Die vorhandenen Quellen zu neuronalen Netzen lassen sich in zwei Kategorien einteilen. Einige sind konzeptuell und mathematisch und enthalten sowohl die Abbildungen, die man typischerweise in Erklärungen neuronaler Netze findet – mit Kreisen, die durch Pfeile verbunden sind –, als auch ausführliche mathematische Erläuterungen der Vorgänge, damit Sie »die Theorie verstehen«. Das wohl beste Beispiel hierfür ist das sehr gute Buch *Deep Learning* von Ian Goodfellow et al. (MIT Press).

Andere Quellen enthalten dicht gepackte Codeblöcke, die einen mit der Zeit abnehmenden Abweichungswert und damit ein neuronales Netz beim »Lernen« zu zeigen scheinen. So definiert das folgende Beispiel aus der PyTorch-Dokumentation tatsächlich ein einfaches neuronales Netz und trainiert es mit zufällig erzeugten Daten:

```
# N is batch size; D_in is input dimension;
# H is hidden dimension; D_out is output dimension.
N, D_in, H, D_out = 64, 1000, 100, 10

# Create random input and output data
x = torch.randn(N, D_in, device=device, dtype=dtype)
y = torch.randn(N, D_out, device=device, dtype=dtype)

# Randomly initialize weights
```

```
w1 = torch.randn(D_in, H, device=device, dtype=dtype)
w2 = torch.randn(H, D_out, device=device, dtype=dtype)

learning_rate = 1e-6
for t in range(500):
    # Forward pass: compute predicted y
    h = x.mm(w1)
    h_relu = h.clamp(min=0)
    y_pred = h_relu.mm(w2)

    # Compute and print loss
    loss = (y_pred - y).pow(2).sum().item()
    print(t, loss)

    # Backprop to compute gradients of w1 and w2 with respect to loss
    grad_y_pred = 2.0 * (y_pred - y)
    grad_w2 = h_relu.t().mm(grad_y_pred)
    grad_h_relu = grad_y_pred.mm(w2.t())
    grad_h = grad_h_relu.clone()
    grad_h[h < 0] = 0
    grad_w1 = x.t().mm(grad_h)

    # Update weights using gradient descent
    w1 -= learning_rate * grad_w1
    w2 -= learning_rate * grad_w2
```

Solche Erklärungen geben natürlich kaum einen Einblick in das, was wirklich passiert – also die zugrunde liegenden mathematischen Prinzipien, die einzelnen Bestandteile des hier gezeigten neuronalen Netzes, ihre Zusammenarbeit und so weiter.[1]

Was *wäre* also eine gute Erklärung der Bestandteile neuronaler Netze? Um eine Antwort zu finden, hilft es, sich anzusehen, wie andere Konzepte der Computerwissenschaft erklärt werden: Wenn Sie etwas über Sortieralgorithmen lernen möchten, gibt es beispielsweise Bücher mit folgendem Inhalt:

- Eine Erklärung des Algorithmus in verständlicher Sprache.
- Eine visuelle Darstellung der Funktionsweise des Algorithmus, so wie Sie es in einem Coding-Interview auf einem Whiteboard notieren würden.
- Eine mathematische Erklärung der Funktionsweise des Algorithmus.[2]
- Pseudocode, der den Algorithmus implementiert.

Diese Elemente findet man in Erklärungen neuronaler Netze selten bis nie gemeinsam, obwohl es mir offensichtlich erscheint, dass eine vollständige Erklärung neu-

1 Aus Fairnessgründen müssen wir sagen, dass dieses Beispiel für die PyTorch-Bibliothek eigentlich für diejenigen gedacht war, die neuronale Netze bereits verstehen, aber nicht als lehrreiche Anleitung. Tatsächlich folgen aber viele Tutorials diesem Stil und zeigen nur den Code mit ein paar kurzen Erläuterungen.

2 Im Fall von Sortieralgorithmen geht es speziell darum, warum der Algorithmus mit einer korrekt sortierten Liste beendet wird.

ronaler Netze auf diese Weise vorgenommen werden sollte. Das Buch ist also der Versuch, diese Lücke zu füllen.

Um neuronale Netze zu verstehen, braucht man mehr als ein Gedankenmodell

Ich bin kein Wissenschaftler und habe auch keinen Doktortitel. Aber ich habe Data Science unterrichtet: Ich habe bei einigen »Data Science Bootcamps« für ein Unternehmen namens Metis unterrichtet und bin ein Jahr lang für Metis um die Welt gereist. Während dieser Zeit führte ich ein- bis fünftätige Workshops für Unternehmen verschiedener Branchen durch, in denen ich den Mitarbeitern maschinelles Lernen und die Grundprinzipien der Softwareentwicklung beibrachte. Ich habe das Unterrichten immer geliebt. Dabei war ich von der Frage fasziniert, wie man technische Konzepte am besten vermitteln kann.

In jüngerer Zeit lag mein Fokus hauptsächlich auf Konzepten des maschinellen Lernens und der Statistik. Bei neuronalen Netzen scheint mir die größte Herausforderung die Vermittlung des richtigen »mentalen Modells« dessen zu sein, was ein neuronales Netz ausmacht. Das gilt besonders, weil man für das Verständnis neuronaler Netze nicht nur ein, sondern *mehrere* Gedankenmodelle braucht, die jeweils verschiedene (aber wichtige) Aspekte der Funktionsweise beleuchten. Zur Illustration sehen Sie unten vier korrekte Antworten auf die Frage »Was ist ein neuronales Netz?«:

- Ein neuronales Netz ist eine mathematische Funktion, die Eingaben übernimmt und Ausgaben erzeugt.
- Ein neuronales Netz ist ein Rechengraph, durch den mehrdimensionale Arrays geleitet werden.
- Ein neuronales Netz besteht aus »Schichten« (»Layers«[3]), die man sich jeweils als eine bestimmte Anzahl von »Neuronen« vorstellen kann.
- Ein neuronales Netz ist ein universeller Funktionsapproximator, der theoretisch die Lösung eines beliebigen Problems des überwachten Lernens darstellt.

Viele von Ihnen kennen diese Definitionen (oder zumindest ein paar davon) und haben bereits ein Grundverständnis ihrer Bedeutung und der Implikationen für die Funktionsweise neuronaler Netze. Für ein umfassendes Verständnis müssen wir jedoch *alle* diese Definitionen verstehen und ihre Verbindungen erkennen können. Worin besteht die gedankliche Verbindung zwischen einem neuronalen Netz als Rechengraph und dem Konzept der »Schichten«?

3 Anm. d. Ü.: Wir werden die Begriffe »Schicht« und »Layer« in diesem Buch synonym verwenden. In manchen Fällen bezeichnet »Layer« gleichzeitig die Schicht selbst und den für ihre Implementierung nötigen Code.

Um diese Zusammenhänge zu verdeutlichen, werden wir die Konzepte von Grund auf in Python implementieren und miteinander verknüpfen. Sie erstellen funktionierende neuronale Netze, die Sie direkt auf Ihrem Laptop trainieren können. Auch wenn wir einen Großteil unserer Zeit mit Implementierungsdetails verbringen werden, geht es bei der Implementierung dieser Modelle in Python darum, die *Konzepte zu verfestigen* und unser *Verständnis davon zu präzisieren*. Was wir hier *nicht* wollen, ist, möglichst knappen Code schreiben oder eine besonders leistungsstarke neuronale Netzwerkbibliothek zu programmieren.

Ich möchte, dass Sie nach dem Lesen dieses Buchs ein solides Verständnis aller dieser mentalen Modelle haben (und was das für die *Implementierung* neuronaler Netze bedeutet). Das soll Ihnen das Lernen verwandter Konzepte und die Durchführung weiterer Projekte in diesem Bereich erleichtern.

Kapitelstruktur

Die ersten drei Kapitel sind die wichtigsten und könnten jeweils selbst ein eigenes Buch füllen.

1. In Kapitel 1, *Grundbausteine*, zeige ich Ihnen, wie mathematische Funktionen zu einer Folge von Operationen verkettet werden können, um einen Rechengraphen zu erstellen. Außerdem erfahren Sie, wie wir anhand dieser Darstellung die Ableitungen der Ausgaben dieser Funktionen bezogen auf die Eingaben berechnen können. Hierfür verwenden wir die Kettenregel aus der Differenzialrechnung. Am Ende des Kapitels stelle ich eine sehr wichtige Operation vor: die Matrizenmultiplikation. Ich zeige, wie sie als mathematische Funktion dargestellt werden kann, während es trotzdem möglich ist, die für das Deep Learning nötigen Ableitungen zu berechnen.
2. In Kapitel 2, *Erste Modelle*, wenden wir die Bausteine aus Kapitel 1 direkt an, um Modelle zu erstellen und zu trainieren, die ein Problem aus der richtigen Welt lösen. Genauer gesagt, verwenden wir sie, um Modelle für lineare Regression und neuronale Netze zu entwickeln, mit denen anhand echter Daten Hauspreise vorhergesagt werden können. Ich zeige Ihnen, dass neuronale Netze leistungsfähiger sind als die lineare Regression, und gebe Ihnen einige Anhaltspunkte für den Grund. Der »First-Principles-Ansatz« (der grundbegriffbasierte Ansatz) für die Erstellung der Modelle in diesem Kapitel soll Ihnen einen guten Eindruck von der Funktionsweise neuronaler Netze vermitteln. Er zeigt aber auch die begrenzten Fähigkeiten eines schrittweisen und nur auf Grundbegriffen basierenden Ansatzes. Dies ist die Motivation für Kapitel 3.
3. In Kapitel 3, *Deep Learning von Grund auf*, verwenden wir die Bausteine aus dem grundbegriffbasierten Ansatz der ersten beiden Kapitel, um allgemeinere Komponenten zu erstellen, aus denen alle Deep-Learning-Modelle bestehen: `Layer` (Schicht), `Model` (Modell), `Optimizer` und so weiter. Wir beenden dieses Kapitel, indem wir ein Deep-Learning-Modell von Grund auf mit dem Daten-

satz aus Kapitel 2 trainieren und zeigen, dass es leistungsfähiger ist als unser einfaches neuronales Netz.

4. Es gibt einige theoretische Garantien dafür, dass ein neuronales Netz einer bestimmten Architektur tatsächlich eine gute Lösung für einen gegebenen Datensatz findet, wenn es mit den in diesem Buch verwendeten Standardtrainingsmethoden trainiert wird. In Kapitel 4, *Techniken zur Verbesserung des Trainings*, behandeln wir die häufigsten »Trainingstricks« zur Erhöhung der Wahrscheinlichkeit, dass ein neuronales Netz eine gute Lösung findet. Nach Möglichkeit geben wir Ihnen mathematische Hinweise darauf, warum dies funktioniert.
5. In Kapitel 5, *CNNs – Faltungsbasierte neuronale Netze*, behandle ich die Grundideen der *Convolutional Neural Networks* (»faltende« neuronale Netze, CNNs), die auf die Interpretation von Bilddaten spezialisiert sind. Hierzu gibt es bereits eine Vielzahl von Erklärungen. Daher konzentriere ich mich auf die absolut wichtigen Grundlagen von CNNs und ihre Unterschiede zu regulären neuronalen Netzen. Insbesondere gehe ich darauf ein, wie es bei CNNs dazu kommt, dass jede Neuronenschicht in »Feature Maps« strukturiert ist. Ich zeige, wie diese Schichten über Faltungsfilter (Convolutional Filters) verbunden sind. Wie die regulären Schichten eines neuronalen Netzes werden wir auch die Faltungsschichten (Convolutional Layers) von Grund auf programmieren, um ihre Funktionsweise besser zu verstehen.
6. In den ersten fünf Kapiteln haben wir eine Miniaturbibliothek für neuronale Netze erstellt, die neurale Netze als eine Reihe von Schichten (`Layer`) definiert, die ihrerseits aus einer Reihe von Operationen bestehen. Diese leiten die Eingaben weiter und schicken Gradienten zurück. In der Praxis werden neuronale Netze jedoch meist anders implementiert: Sie verwenden die sogenannte *automatische Differenzierung* (Automatic Differentiation). Zu Beginn von Kapitel 6, *RNNs – Rekurrente neuronale Netze*, gebe ich Ihnen eine kurze Einführung in das automatische Differenzieren und verwende diese Technik, um das Hauptthema dieses Kapitels vorzustellen: Rekurrente neuronale Netze (RNNs). Diese Architektur wird typischerweise verwendet, um Daten zu verstehen, deren Datenpunkte als Folgen (sequenziell) auftreten, zum Beispiel Zeitreihen oder natürliche Sprachdaten. Ich erkläre die Funktionsweise »normaler« RNNs sowie zweier Varianten: *GRUs* und *LSTMs* (die wir natürlich ebenfalls von Grund auf implementieren). Dabei zeige ich, welche Dinge alle RNN-Varianten *gemeinsam* haben und welche *Unterschiede* es gibt.
7. Zum Abschluss zeige ich Ihnen in Kapitel 7, *PyTorch*, wie die in den Kapiteln 1 bis 6 behandelten Dinge mithilfe der hochperformanten Open-Source-Bibliothek für neuronale Netze PyTorch implementiert werden können. Um mehr über neuronale Netze zu erfahren, ist es unabdingbar, ein Framework zu lernen. Ohne ein solides Verständnis der Funktionsweise neuronaler Netze werden Sie auf lange Sicht aber auch mit dem besten Framework nicht weiter-

kommen. Ich möchte Ihnen mit diesem Buch helfen, hochperformante neuronale Netze zu schreiben (indem ich Ihnen PyTorch beibringe), und Sie gleichzeitig auf das langfristige Lernen und das erfolgreiche Anwenden des Wissen vorbereiten (indem ich Ihnen vorher die Grundlagen vermittle). Wir beschließen das Buch mit einem kurzen Blick auf die Verwendung neuronaler Netze für das unüberwachte Lernen.

Mein Ziel war es, das Buch zu schreiben, das mir gefehlt hat, als ich vor einigen Jahren damit begann, mich mit diesem Thema zu befassen. Ich hoffe, dieses Buch hilft Ihnen. Auf geht's!

In diesem Buch verwendete Konventionen

Die folgenden typografischen Konventionen werden in diesem Buch verwendet:

Kursiv
: Kennzeichnet neue Begriffe, URLs, E-Mail-Adressen, Dateinamen und Dateiendungen.

`Konstante Zeichenbreite`
: Wird für Programmlistings und für Programmelemente in Textabschnitten wie Namen von Variablen und Funktionen, Datenbanken, Datentypen und Umgebungsvariablen sowie für Anweisungen und Schlüsselwörter verwendet.

`Konstante Zeichenbreite, fett`
: Kennzeichnet Befehle oder anderen Text, den der Nutzer wörtlich eingeben sollte.

`Konstante Zeichenbreite, kursiv`
: Kennzeichnet Text, den der Nutzer je nach Kontext durch entsprechende Werte ersetzen sollte.

Der Satz des Pythagoras lautet: $a^2 + b^2 = c^2$.

Dieses Symbol steht für einen allgemeinen Hinweis.

Codebeispiele

Weiterführendes Material (Codebeispiele, Übungen etc.) steht im GitHub-Repository zu diesem Buch (*https://oreil.ly/deep-learning-github*) zum Download bereit.

Dieses Buch dient dazu, Ihnen beim Erledigen Ihrer Arbeit zu helfen. Im Allgemeinen dürfen Sie die Codebeispiele aus diesem Buch in Ihren eigenen Programmen und der dazugehörigen Dokumentation verwenden. Sie müssen uns dazu nicht um Erlaubnis fragen, solange Sie nicht einen beträchtlichen Teil des Codes reproduzie-

ren. Beispielsweise benötigen Sie keine Erlaubnis, um ein Programm zu schreiben, in dem mehrere Codefragmente aus diesem Buch vorkommen. Wollen Sie dagegen eine CD-ROM mit Beispielen aus Büchern von O'Reilly verkaufen oder verteilen, benötigen Sie eine Erlaubnis. Eine Frage zu beantworten, indem Sie aus diesem Buch zitieren und ein Codebeispiel wiedergeben, benötigt keine Erlaubnis. Eine beträchtliche Menge Beispielcode aus diesem Buch in die Dokumentation Ihres Produkts aufzunehmen, bedarf hingegen einer Erlaubnis.

Wir freuen uns über Zitate, verlangen diese aber nicht. Ein Zitat enthält Titel, Autor, Verlag und ISBN, beispielsweise: *Deep Learning – Grundlagen und Implementierung* von Seth Weidman (O'Reilly). Copyright 2019 Seth Weidman, ISBN 978-3-96009-136-3.«

Wenn Sie glauben, dass Ihre Verwendung von Codebeispielen über die übliche Nutzung hinausgeht oder außerhalb der oben vorgestellten Nutzungsbedingungen liegt, kontaktieren Sie uns bitte unter *kommentar@oreilly.de*.

Danksagungen des Autors

Ich möchte mich bei meiner Lektorin Melissa Potter und dem Team bei O'Reilly bedanken für ihre unzähligen Rückmeldungen und Antworten auf meine vielen Fragen während des gesamten Prozesses.

Ein besonderes Dankeschön geht an die vielen Menschen, deren Arbeit, die technischen Konzepte des maschinellen Lernens einem breiteren Publikum zugänglich zu machen, mich direkt beeinflusst habt. Ich hatte das Glück, einige dieser Menschen persönlich kennenlernen zu dürfen: In zufälliger Reihenfolge sind dies Brandon Rohrer, Joel Grus, Jeremy Watt und Andrew Trask.

Ich möchte mich außerdem bei meinem Chef bei Metis bedanken und meinem Director bei Facebook. Beide haben mir sehr dabei geholfen, Zeit zu finden für die Arbeit an diesem Projekt.

Würdigung und besonderer Dank gebührt Mat Leonhard, der für kurze Zeit mein Koautor war, bevor wir uns entschieden, eigene Wege zu gehen. Mat half mir, den Code für die Mini-Bibliothek zu diesem Buch zu organisieren – `lincoln` –, und gab mir Feedback zu den ersten beiden Kapiteln, wobei er für große Abschnitte dieser Kapitel seine eigenen Versionen schrieb.

Schließlich möchte ich mich noch bei meinen Freunden Eva und John bedanken, die mich inspiriert und dazu motiviert haben, ins kalte Wasser zu springen und tatsächlich mit dem Schreiben zu beginnen. Ich bedanke mich außerdem bei meinen vielen Freunden in San Francisco, die es ausgehalten haben, dass ich ständig Sorgen und Vorbehalte bezüglich dieses Buchs hatte und für viele Monate kaum für gemeinsame Unternehmungen verfügbar war. Sie haben mich unerschütterlich unterstützt, als ich sie brauchte.