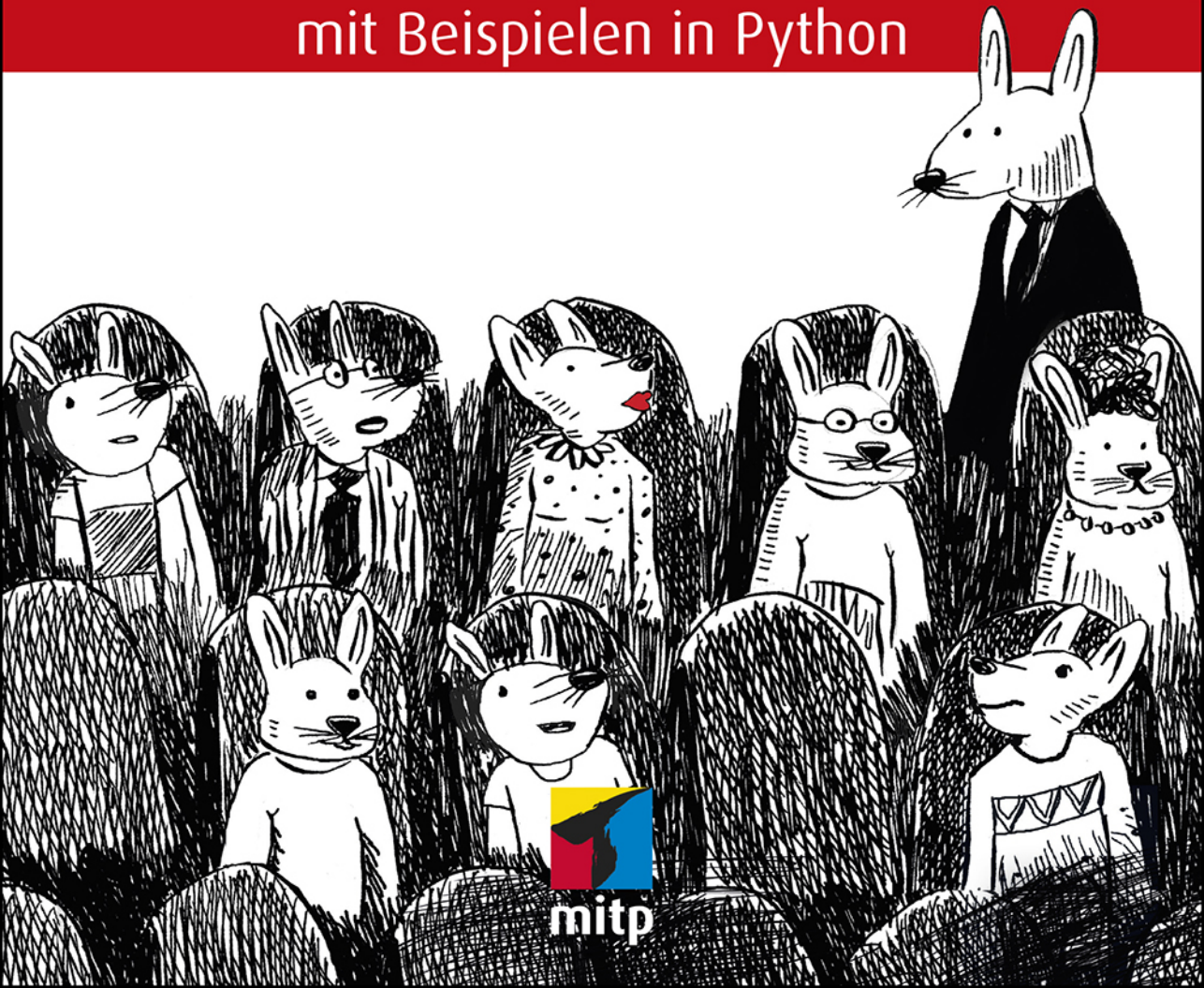


ANDREW W. TRASK

# NEURONALE NETZE UND DEEP LEARNING KAPIEREN

DER EINFACHE PRAXISEINSTIEG

mit Beispielen in Python



# Inhaltsverzeichnis



<b>Vorwort</b> .....	13
<b>Einleitung</b> .....	15
<b>Über den Autor</b> .....	17
<b>Danksagung</b> .....	17
<b>1 Deep Learning: Weshalb man sich damit befassen sollte</b> .....	19
1.1 Willkommen bei »Deep Learning kapieren« .....	19
1.2 Weshalb du dich mit Deep Learning befassen solltest .....	19
1.3 Ist es schwierig, Deep Learning zu verstehen? .....	21
1.4 Warum du dieses Buch lesen solltest .....	21
1.5 Was du brauchst, um loszulegen .....	23
1.6 Python-Kenntnisse sind nützlich .....	24
1.7 Zusammenfassung .....	24
<b>2 Grundlegende Konzepte: Wie lernen Maschinen?</b> .....	25
2.1 Was ist Deep Learning? .....	25
2.2 Was ist Machine Learning? .....	26
2.3 Überwachtes Machine Learning .....	27
2.4 Unüberwachtes Machine Learning .....	28
2.5 Parametrisches und nichtparametrisches Lernen .....	29
2.6 Überwachtes parametrisches Lernen .....	30
2.7 Unüberwachtes parametrisches Lernen .....	33
2.8 Nichtparametrisches Lernen .....	34
2.9 Zusammenfassung .....	35

<b>3</b>	<b>Vorhersage mit neuronalen Netzen: Forward Propagation</b> . . . . .	<b>37</b>
3.1	Vorhersage . . . . .	37
3.2	Ein einfaches neuronales Netz trifft eine Vorhersage. . . . .	39
3.3	Was ist ein neuronales Netz? . . . . .	40
3.4	Wie funktioniert das neuronale Netz? . . . . .	41
3.5	Eine Vorhersage mit mehreren Eingaben treffen . . . . .	44
3.6	Mehrere Eingaben: Wie verhält sich das neuronale Netz? . . . . .	45
3.7	Mehrere Eingaben: vollständiger ausführbarer Code . . . . .	51
3.8	Eine Vorhersage mit mehreren Ausgaben treffen . . . . .	52
3.9	Vorhersagen mit mehreren Eingaben und mehreren Ausgaben treffen . . . . .	54
3.10	Mehrere Ein- und Ausgaben: Wie funktioniert das? . . . . .	56
3.11	Vorhersagen über Vorhersagen . . . . .	58
3.12	Kurzeinführung in NumPy. . . . .	60
3.13	Zusammenfassung . . . . .	64
<b>4</b>	<b>Lernen in neuronalen Netzen: Gradientenabstieg</b> . . . . .	<b>65</b>
4.1	Vorhersagen, Vergleichen und Erlernen . . . . .	65
4.2	Vergleichen. . . . .	66
4.3	Erlernen . . . . .	66
4.4	Vergleichen: Trifft das Netz gute Vorhersagen? . . . . .	67
4.5	Warum Fehler messen? . . . . .	68
4.6	Was ist die einfachste Form des Lernens in neuronalen Netzen? . . . . .	69
4.7	Hot und Cold Learning . . . . .	71
4.8	Hot und Cold Learning: Eigenschaften . . . . .	72
4.9	Richtung und Betrag anhand des Fehlers berechnen . . . . .	73
4.10	Eine Iteration des Gradientenabstiegs . . . . .	76
4.11	Lernen bedeutet nur, den Fehler zu verringern . . . . .	78
4.12	Mehrere Schritte des Lernens . . . . .	80
4.13	Weshalb funktioniert das? Was ist <code>weight_delta</code> eigentlich? . . . . .	82
4.14	Tunnelblick auf ein Konzept. . . . .	84
4.15	Eine Kiste, aus der zwei Stangen ragen . . . . .	85
4.16	Ableitungen: Zweiter Ansatz . . . . .	86
4.17	Was du wirklich wissen musst . . . . .	87
4.18	Was man eigentlich nicht wissen muss . . . . .	88
4.19	Wie man Ableitungen zum Lernen verwendet . . . . .	89
4.20	Wirkt das vertraut? . . . . .	90
4.21	Den Gradientenabstieg stören . . . . .	91
4.22	Visualisierung der übermäßigen Korrektur. . . . .	92

4.23	Divergenz . . . . .	93
4.24	Einführung von alpha . . . . .	94
4.25	Alpha im Code . . . . .	95
4.26	Auswendig lernen . . . . .	96
<b>5</b>	<b>Mehrere Gewichte gleichzeitig erlernen: Generalisierung des Gradientenabstiegs . . . . .</b>	<b>99</b>
5.1	Lernen durch Gradientenabstieg mit mehreren Eingaben . . . . .	99
5.2	Gradientenabstieg mit mehreren Eingaben erklärt . . . . .	102
5.3	Mehrere Lernschritte . . . . .	106
5.4	Einfrieren eines Gewichts: Was bewirkt das? . . . . .	109
5.5	Lernen mittels Gradientenabstieg mit mehreren Ausgaben . . . . .	111
5.6	Lernen mittels Gradientenabstieg mit mehreren Eingaben und mehreren Ausgaben . . . . .	114
5.7	Was erlernen die Gewichte? . . . . .	116
5.8	Visualisierung der Werte von Gewichten . . . . .	118
5.9	Visualisierung von Skalarprodukten (gewichtete Summen) . . . . .	119
5.10	Zusammenfassung . . . . .	120
<b>6</b>	<b>Das erste tiefe neuronale Netz: Einführung in Backpropagation . . .</b>	<b>121</b>
6.1	Das Ampelproblem . . . . .	121
6.2	Vorbereitung der Daten . . . . .	123
6.3	Matrizen . . . . .	124
6.4	Erstellen von Matrizen in Python . . . . .	128
6.5	Erstellen eines neuronalen Netzes . . . . .	129
6.6	Erlernen der gesamten Datenmenge . . . . .	130
6.7	Vollständiger, Batch- und stochastischer Gradientenabstieg . . . . .	131
6.8	Neuronale Netze erlernen eine Korrelation . . . . .	132
6.9	Druck nach oben und unten . . . . .	133
6.10	Grenzfall: Überanpassung . . . . .	135
6.11	Grenzfall: Widersprüchliche Druckkräfte . . . . .	137
6.12	Erlernen indirekter Korrelation . . . . .	139
6.13	Korrelation erzeugen . . . . .	140
6.14	Stapeln neuronaler Netze: Überblick . . . . .	141
6.15	Backpropagation: Fehlerattribuierung . . . . .	142
6.16	Backpropagation: Weshalb funktioniert das? . . . . .	143
6.17	Linear vs. nichtlinear . . . . .	144
6.18	Weshalb das neuronale Netz noch nicht funktioniert . . . . .	145
6.19	Bedingte Korrelation . . . . .	146

6.20	Eine kurze Pause . . . . .	148
6.21	Das erste tiefe neuronale Netz . . . . .	148
6.22	Backpropagation im Code . . . . .	150
6.23	Eine Iteration der Backpropagation . . . . .	152
6.24	Alles zusammenbringen . . . . .	154
6.25	Warum sind tiefe Netze von Bedeutung? . . . . .	155
7	<b>Neuronale Netze abbilden: im Kopf und auf Papier</b> . . . . .	157
7.1	Zeit für Vereinfachungen . . . . .	157
7.2	Korrelationszusammenfassung . . . . .	158
7.3	Die alte, zu komplizierte Visualisierung . . . . .	160
7.4	Die vereinfachte Visualisierung . . . . .	161
7.5	Weitere Vereinfachung . . . . .	162
7.6	Ein Netz bei der Vorhersage beobachten . . . . .	164
7.7	Visualisierung mit Buchstaben statt Bildern . . . . .	165
7.8	Variablen verknüpfen . . . . .	166
7.9	Alles zusammen. . . . .	166
7.10	Die Bedeutung von Visualisierungstools . . . . .	167
8	<b>Signale erlernen, Rauschen ignorieren: Einführung in Regularisierung und Batching</b> . . . . .	169
8.1	Ein dreischichtiges Netz mit dem MNIST-Datensatz . . . . .	169
8.2	Das war einfach . . . . .	171
8.3	Auswendiglernen vs. Generalisierung . . . . .	173
8.4	Überanpassung in neuronalen Netzen . . . . .	174
8.5	Wie Überanpassung entsteht . . . . .	175
8.6	Die einfachste Regularisierung: früher Abbruch . . . . .	176
8.7	Der Industriestandard: das Dropout-Verfahren . . . . .	177
8.8	Weshalb das Dropout-Verfahren funktioniert: Ensembles. . . . .	178
8.9	Das Dropout-Verfahren im Code . . . . .	179
8.10	Bewertung des Dropout-Verfahrens mit dem MNIST-Datensatz . . . . .	182
8.11	Batch-Gradientenabstieg . . . . .	183
8.12	Zusammenfassung . . . . .	186
9	<b>Modellierung von Wahrscheinlichkeiten und Nichtlinearitäten: Aktivierungsfunktionen</b> . . . . .	187
9.1	Was ist eine Aktivierungsfunktion? . . . . .	187
9.2	Standardaktivierungsfunktionen der verdeckten Schicht. . . . .	191
9.3	Standardaktivierungsfunktionen der Ausgabeschicht . . . . .	192

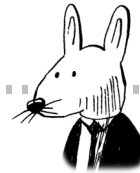
9.4	Das grundlegende Problem: Eingaben ähneln einander . . . . .	194
9.5	softmax-Berechnung . . . . .	195
9.6	Anleitung für die Nutzung von Aktivierungsfunktionen . . . . .	197
9.7	Multiplikation von delta mit der Steigung . . . . .	199
9.8	Ausgabe in Steigung umwandeln (Ableitung) . . . . .	200
9.9	Verbesserung des MNIST-Netzes . . . . .	201
<b>10</b>	<b>Einführung in Convolutional Neural Networks . . . . .</b>	<b>205</b>
10.1	Wiederverwendung von Gewichten an verschiedenen Stellen . . . . .	205
10.2	Die Faltungsschicht. . . . .	206
10.3	Eine einfache Implementierung in NumPy . . . . .	209
10.4	Zusammenfassung . . . . .	214
<b>11</b>	<b>Neuronale Netze, die Sprache verstehen:</b>	
	<b>Knecht – Mann + Frau == ? . . . . .</b>	<b>217</b>
11.1	Was bedeutet es, Sprache zu verstehen? . . . . .	217
11.2	Verarbeitung natürlicher Sprache . . . . .	218
11.3	Überwachte Verarbeitung natürlicher Sprache. . . . .	219
11.4	Die IMDB-Filmbewertungsdatenbank. . . . .	220
11.5	Wortkorrelationen in Eingabedaten erfassen . . . . .	221
11.6	Vorhersage von Filmbewertungen . . . . .	223
11.7	Kurz vorgestellt: Embedding-Schichten. . . . .	224
11.8	Interpretation der Ausgabe . . . . .	227
11.9	Architektur neuronaler Netze. . . . .	228
11.10	Wort-Embeddings vergleichen . . . . .	230
11.11	Welche Bedeutung hat ein Neuron? . . . . .	232
11.12	Ausfüllen der Lücke . . . . .	233
11.13	Bedeutung ergibt sich aus dem Verlust. . . . .	236
11.14	Knecht – Mann + Frau $\approx$ Magd . . . . .	239
11.15	Wortanalogien . . . . .	240
11.16	Zusammenfassung . . . . .	241
<b>12</b>	<b>Rekurrente Schichten für Daten variabler Größe . . . . .</b>	<b>243</b>
12.1	Die Herausforderungen von variabler Größe . . . . .	243
12.2	Spielen Vergleiche tatsächlich eine Rolle? . . . . .	244
12.3	Die erstaunliche Leistungsstärke gemittelter Wortvektoren . . . . .	246
12.4	Wie werden Informationen in diesen Embeddings gespeichert? . . . . .	247
12.5	Wie verwendet ein neuronales Netz Embeddings? . . . . .	248
12.6	Die Beschränkungen von Bag-of-words-Vektoren . . . . .	249

12.7	Verwendung von Einheitsmatrizen zum Summieren von Wort-Embeddings . . . . .	251
12.8	Matrizen, die überhaupt nichts verändern. . . . .	252
12.9	Erlernen der Übergangsmatrizen. . . . .	253
12.10	Lernen, nützliche Satzvektoren zu erzeugen. . . . .	254
12.11	Forward Propagation in Python . . . . .	256
12.12	Wie funktioniert die Backpropagation? . . . . .	257
12.13	Training durchführen . . . . .	258
12.14	Einrichtung. . . . .	259
12.15	Forward Propagation beliebiger Größe . . . . .	261
12.16	Backpropagation beliebiger Größe . . . . .	262
12.17	Aktualisierung von Gewichten beliebiger Größe . . . . .	263
12.18	Ausführung und Analyse der Ausgabe . . . . .	264
12.19	Zusammenfassung . . . . .	267
<b>13</b>	<b>Automatische Optimierung: Entwicklung eines Deep-Learning-Frameworks . . . . .</b>	<b>269</b>
13.1	Was ist ein Deep-Learning-Framework? . . . . .	269
13.2	Einführung in Tensoren . . . . .	271
13.3	Einführung in Automatic Gradient Computation (Autograd) . . . . .	272
13.4	Eine kurze Bestandsaufnahme . . . . .	274
13.5	Mehrfach verwendete Tensoren . . . . .	275
13.6	Autograd um mehrmals verwendbare Tensoren erweitern . . . . .	276
13.7	Wie funktioniert Backpropagation durch Addition? . . . . .	279
13.8	Negation hinzufügen. . . . .	280
13.9	Weitere Funktionen hinzufügen . . . . .	281
13.10	Autograd zum Trainieren eines neuronalen Netzes verwenden . . . . .	286
13.11	Automatische Optimierung hinzufügen . . . . .	288
13.12	Schichttypen hinzufügen . . . . .	290
13.13	Schichten, die weitere Schichten enthalten . . . . .	291
13.14	Verlustfunktions-Schichten. . . . .	292
13.15	Erlernen eines Frameworks . . . . .	293
13.16	Nichtlineare Schichten . . . . .	294
13.17	Die Embedding-Schicht. . . . .	296
13.18	Indizierung zu Autograd hinzufügen . . . . .	297
13.19	Zurück zur Embedding-Schicht . . . . .	299
13.20	Die Kreuzentropie-Schicht . . . . .	300
13.21	Die rekurrente Schicht eines neuronalen Netzes . . . . .	302
13.22	Zusammenfassung . . . . .	307

<b>14</b>	<b>Lernen, wie Shakespeare zu schreiben:</b>	
	<b>Long Short-Term Memory</b> . . . . .	309
14.1	Zeichenbasierte Sprachmodellierung . . . . .	309
14.2	Die Notwendigkeit einer verkürzten Backpropagation. . . . .	310
14.3	Truncated Backpropagation . . . . .	312
14.4	Eine Stichprobe von der Ausgabe. . . . .	315
14.5	Verschwindende und explodierende Gradienten . . . . .	317
14.6	Ein Testbeispiel für Backpropagation in RNNs. . . . .	318
14.7	LSTM-Zellen . . . . .	320
14.8	Veranschaulichung von LSTM-Gates. . . . .	321
14.9	Die LSTM-Schicht . . . . .	322
14.10	Erweiterung des zeichenbasierten Sprachmodells . . . . .	324
14.11	Training des zeichenbasierten LSTM-Sprachmodells . . . . .	325
14.12	Optimierung des zeichenbasierten LSTM-Sprachmodells. . . . .	327
14.13	Zusammenfassung . . . . .	328
<b>15</b>	<b>Deep Learning mit unbekanntem Daten: Federated Learning</b> . . . . .	329
15.1	Das Problem der Privatsphäre beim Deep Learning. . . . .	329
15.2	Federated Learning . . . . .	330
15.3	Spam-Erkennung erlernen . . . . .	332
15.4	Federation . . . . .	334
15.5	Federated Learning hacken. . . . .	336
15.6	Sichere Aggregation . . . . .	337
15.7	Homomorphe Verschlüsselung . . . . .	338
15.8	Homomorph verschlüsseltes Federated Learning . . . . .	339
15.9	Zusammenfassung . . . . .	341
<b>16</b>	<b>Wie geht es weiter? Ein kurzer Leitfaden</b> . . . . .	343
16.1	Glückwunsch! . . . . .	343
16.2	Schritt 1: Lerne PyTorch . . . . .	344
16.3	Schritt 2: Nimm an einem weiteren Deep-Learning-Kurs teil . . . . .	344
16.4	Schritt 3: Lies ein mathematisch anspruchsvolles Lehrbuch . . . . .	345
16.5	Schritt 4: Schreibe ein Blog und unterrichte Deep Learning. . . . .	345
16.6	Schritt 5: Twitter . . . . .	347
16.7	Schritt 6: Verweise auf wissenschaftliche Arbeiten . . . . .	347
16.8	Schritt 7: Verschaffe dir Zugriff auf eine GPU (oder mehrere). . . . .	347
16.9	Schritt 8: Lass dich für praktische Erfahrungen bezahlen . . . . .	348
16.10	Schritt 9: Beteilige dich an einem Open-Source-Projekt . . . . .	348
16.11	Schritt 10: Fördere deine Community vor Ort. . . . .	349
	<b>Stichwortverzeichnis</b> . . . . .	351



# Vorwort



*Deep Learning kapieren* ist das Ergebnis von drei Jahren harter Arbeit. Ich habe mindestens doppelt so viele Seiten geschrieben, wie dieses Buch in der vorliegenden Form aufweist. Ein halbes Dutzend Kapitel mussten drei oder vier Mal völlig neu verfasst werden, bevor sie reif für die Veröffentlichung waren, und zwischenzeitlich kamen wichtige Kapitel hinzu, die ursprünglich gar nicht vorgesehen waren.

Von viel größerer Bedeutung ist allerdings, dass ich frühzeitig zwei Entschlüsse gefasst habe, die dieses Buch einzigartig machen: Bis auf grundlegende Arithmetik sind keine mathematischen Kenntnisse erforderlich, und das Buch beruht nicht auf einer der allgemeinen Bibliotheken, die oftmals verbergen können, was eigentlich vor sich geht. Mit anderen Worten: Jeder kann dieses Buch lesen und verstehen, wie Deep Learning tatsächlich funktioniert. Um das zu erreichen, musste ich mir neue Methoden ausdenken, um die grundlegenden Konzepte und Verfahren zu beschreiben, ohne auf höhere Mathematik zurückzugreifen oder ausgeklügelten Code zu verwenden, den jemand anderes geschrieben hat.

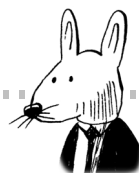
Mein Ziel beim Schreiben des Buchs war, den praktischen Einstieg in Deep Learning so einfach wie möglich zu machen. Du liest nicht einfach nur über die Theorie des Deep Learnings, du entdeckst es selbst. Um dich dabei zu unterstützen, habe ich eine Menge Code geschrieben und mir große Mühe gegeben, ihn in der richtigen Reihenfolge zu erklären, damit die für die lauffähigen Demos erforderlichen Codeschnipsel einen Sinn ergeben.

Zusammen mit der Theorie, dem Code und den Beispielen, die du in diesem Buch erkunden wirst, ermöglicht dir dieses Wissen, viel schneller zu experimentieren. Du wirst rasch Erfolge erzielen, dir stehen bessere Karriereemöglichkeiten offen und du wirst fortgeschrittenere Deep-Learning-Konzepte schneller verstehen.

In den vergangenen drei Jahren habe ich nicht nur dieses Buch geschrieben, sondern auch an einem Doktorandenprogramm in Oxford teilgenommen, bin dem Team bei Google beigetreten und habe an der Entwicklung von OpenMined mitgewirkt, einer dezentralisierten Plattform für künstliche Intelligenz. Dieses Buch fasst also Jahre des Denkens, des Lernens und des Lehrens zusammen.

Es gibt eine Vielzahl von Ressourcen, um Deep Learning zu lernen. Es freut mich, dass du dich für diese hier entschieden hast.

# Einleitung



Dieses Buch soll dir die Grundlagen des Deep Learnings vermitteln, damit du eines der großen Deep-Learning-Frameworks verwenden kannst. Wir konzentrieren uns zunächst auf elementare neuronale Netze und werfen anschließend einen Blick auf komplexere Schichten und Architekturen.

## Wer sollte dieses Buch lesen?

Ich habe das Buch bewusst so geschrieben, dass es möglichst leicht verständlich ist und keine Kenntnisse der linearen Algebra, der Infinitesimalrechnung, konvexer Optimierungen oder auch nur des Machine Learnings voraussetzt. Alles aus diesen Themenbereichen, was für das Verständnis des Deep Learnings erforderlich ist, wird erklärt, sobald wir uns damit befassen. Wenn du in der Schule mit Mathematik zurechtgekommen bist und schon ein wenig in Python programmiert hast, bist du bereit für dieses Buch.

## Überblick

Das Buch hat 16 Kapitel:

- Kapitel 1 konzentriert sich darauf, weshalb du dich mit Deep Learning befassen solltest und was du für den Anfang brauchst.
- Kapitel 2 untersucht grundlegende Konzepte, wie Machine Learning, parametrische und nichtparametrische Modelle oder überwachtes und unüberwachtes Lernen. Hier wird auch das Paradigma »Vorhersagen, Vergleichen, Erlernen« vorgestellt, das in den nachfolgenden Kapiteln immer wieder vorkommt.
- Kapitel 3 führt vor, wie du einfache Netze dazu verwenden kannst, Vorhersagen zu treffen, und kommt erstmals zu neuronalen Netzen.

- Kapitel 4 zeigt dir, wie du die in Kapitel 3 getroffenen Vorhersagen bewerten kannst und Fehler aufspürst, um im nächsten Schritt Modelle zu trainieren.
- Kapitel 5 konzentriert sich auf das »Erlernen« aus dem Paradigma »Vorhersagen, Vergleichen, Erlernen«. Anhand eines umfassenden Beispiels untersuchen wir den Lernvorgang.
- In Kapitel 6 wirst du dein erstes »tiefes« neuronales Netz erstellen – inklusive Code.
- Kapitel 7 betrachtet neuronale Netze aus der Vogelperspektive und soll deine Vorstellung davon vereinfachen.
- Kapitel 8 stellt Überanpassung, Dropout-Verfahren und Gradientenabstieg vor und zeigt dir, wie du deine Datenmenge in dem gerade erstellten Netz klassifizierst.
- Kapitel 9 erklärt Aktivierungsfunktionen und wie man sie verwendet, wenn man Wahrscheinlichkeiten modelliert.
- Kapitel 10 stellt neuronale Faltungsnetze oder CNNs (*Convolutional Neural Networks*) vor und stellt den Nutzen von Struktur bei der Vermeidung von Überanpassung heraus.
- Kapitel 11 befasst sich mit der Verarbeitung natürlicher Sprache (*Natural Language Processing*, NLP) und erläutert grundlegende Terminologie und Konzepte des Fachgebiets Deep Learning.
- Kapitel 12 erörtert rekurrente neuronale Netze (RNNs), einen modernen Ansatz, der bei Sequenzmodellierungen in fast allen Fachgebieten aktuell und in der Branche mit am weitesten verbreitet ist.
- Kapitel 13 bietet einen Schnellkurs für das Erstellen eines Deep-Learning-Frameworks von Grund auf.
- Kapitel 14 verwendet dein rekurrentes neuronales Netz, um eine schwierigere Aufgabe in Angriff zu nehmen: Sprachmodellierung.
- In Kapitel 15 geht es um den Datenschutz und um grundlegende Konzepte wie Federated Learning, homomorphe Verschlüsselung und weitere Konzepte, die mit Differential Privacy und sicheren Berechnungen, wenn mehrere Parteien beteiligt sind, zu tun haben.
- Kapitel 16 stellt die Tools und Ressourcen bereit, die dir beim weiteren Erkunden des Deep Learnings gute Dienste leisten werden.

## Konventionen und Downloads

Für den im Buch abgedruckten Code wird eine nicht-proportionale Schrift verwendet, um ihn vom Fließtext zu unterscheiden. Einige der Listings enthalten Anmerkungen, die wichtige Konzepte hervorheben.

Die Codebeispiele kannst du unter <https://github.com/iamtrask/grokking-deep-learning> oder unter [www.mitp.de/0015](http://www.mitp.de/0015) herunterladen.

## Über den Autor

Andrew Trask ist Gründungsmitglied von Digital Reasonings Machine Learning Lab, in dem Deep-Learning-Ansätze zur Verarbeitung natürlicher Sprache, Bilderkennung und Audiotranskription erforscht werden. Innerhalb weniger Monate gelang es Andrew und seinen Forschungskollegen, die besten veröffentlichten Ergebnisse bei der Stimmungsanalyse und der automatisierten Zuordnung von Wörtern eines Textes zu Wortgruppen (Part-of-Speech-Tagging) zu übertreffen. Er hat das größte künstliche neuronale Netz mit mehr als 160 Milliarden Parametern trainiert. Die Ergebnisse hat er zusammen mit seinem Koautor auf der internationalen Konferenz über Machine Learning vorgestellt. Sie wurden im *Journal of Machine Learning* veröffentlicht.

Derzeit ist er als Produktmanager für Text- und Audioanalytik bei Digital Reasoning für die Weiterentwicklung der kognitiven Computing-Plattform *Synthesis* verantwortlich, wobei Deep Learning zu den Kernkompetenzen gehört.

## Danksagung

Ich bin allen außerordentlich dankbar, die an der Entstehung des Buchs mitgewirkt haben. An erster Stelle möchte ich mich bei dem tollen Team des amerikanischen Originalverlags Manning bedanken: Bert Bates, der mich lehrte, wie man schreibt, Christina Taylor, die mich drei Jahre lang geduldig zum Fortfahren anhielt, Michael Stephens, dessen Kreativität es dem Buch ermöglichte, schon vor der Veröffentlichung ein Erfolg zu sein, und Marjan Bace, deren Ermunterungen während der ganzen Verzögerungen den Ausschlag gaben.

Das Buch wäre ohne die vielen Beiträge von Lesern per E-Mail, Twitter und GitHub nicht das, was es ist. Ich bin folgenden Personen für ihre Hilfe bei der Verbesserung des Texts und des Codes zu großem Dank verpflichtet: Jascha Swisher, Varun Sudhakar, Francois Chollet, Frederico Vitorino, Cody Hammond, Mauricio Maroto Arrieta, Aleksandar Dragosavljevic, Alan Carter, Frank Hinek, Nicolas Benjamin Hocker, Hank Meisse, Wouter Hibma, Joerg Rosenkranz, Alex Vieira und Charlie Harrington.

Ich danke den Korrektoren, die sich die Zeit genommen haben, das Manuskript während verschiedener Phasen der Entwicklung zu lesen: Alexander A. Myltsev, Amit Lamba, Anand Saha, Andrew Hamor, Cristian Barrientos, Montoya, Eremey Valetov, Gerald Mack, Ian Stirk, Kalyan Reddy, Kamal Raj, Kelvin D. Meeks, Marco Paulo dos Santos Nogueira, Martin Beer, Massimo Ilario, Nancy W. Grady, Peter Hampton, Sebastian Maldonado, Shashank Gupta, Tymoteusz Wołodźko, Kumar Unnikrishnan, Vipul Gupta, Will Fuger und William Wheeler.

Ich danke Mat und Niko bei Udacity, die das Buch in das Nanodegree-Programm aufgenommen haben, was sehr dazu beigetragen hat, es unter angehenden Deep-Learning-Entwicklern bekannt zu machen.

Ich möchte Dr. William Hooper danken, der stets ein offenes Ohr für meine Fragen über Informatik hatte und mich ausnahmsweise an seinem eigentlich schon voll belegten Kurs »Programmieren 1« teilnehmen ließ. Er hat mich dazu inspiriert, eine Laufbahn im Deep Learning zu verfolgen. Ich bin außerordentlich dankbar für all die entgegengebrachte Geduld, die mir sehr geholfen hat.

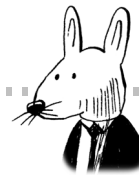
Und schließlich möchte ich meiner Frau dafür danken, dass sie während all der Nächte und Wochenenden, die ich mit der Arbeit am Buch verbracht habe, so viel Geduld mit mir hatte und dass sie den gesamten Text des Buchs mehrmals eigenhändig redigiert hat und sich um die Erstellung und Pflege des Code-Repositorys bei GitHub gekümmert hat.

# Grundlegende Konzepte: Wie lernen Maschinen?

# 2

*In fünf Jahren wird Machine Learning für jeden erfolgreichen Börsengang verantwortlich sein.*

- Eric Schmidt, Vorstandsvorsitzender bei Google, in einer Rede auf der Cloud Computing Platform Conference 2016



## In diesem Kapitel:

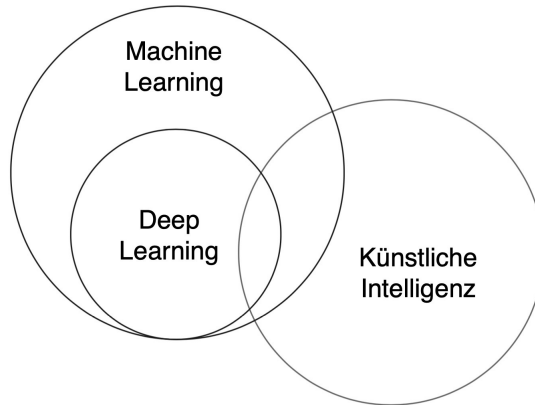
- Deep Learning, Machine Learning und künstliche Intelligenz
- Parametrische und nichtparametrische Modelle
- Überwachtes und unüberwachtes Lernen
- Wie können Maschinen lernen?

## 2.1 Was ist Deep Learning?

### **Deep Learning ist eine Teilmenge der Machine-Learning-Verfahren.**

Deep Learning ist eine Teilmenge des Machine Learnings, das sich der Untersuchung und der Entwicklung von Maschinen widmet, die lernen können (manchmal auch mit dem Ziel, eine allgemeine künstliche Intelligenz zu erlangen).

In der Industrie wird Deep Learning in verschiedenen Bereichen zum Lösen praktischer Aufgaben eingesetzt, etwa bei Computer Vision (Bilder), bei der Verarbeitung natürlicher Sprache (Text) oder bei der automatischen Spracherkennung (Audio). Deep Learning ist also eine Teilmenge der *Verfahren*, die beim Machine Learning zum Einsatz kommen, wobei vor allem künstliche neuronale Netze verwendet werden, die zu einer Klasse von Algorithmen gehören, die mehr oder weniger durch das menschliche Gehirn inspiriert wurden.



Die Abbildung zeigt, dass es beim Deep Learning nicht nur um allgemeine künstliche Intelligenz geht (wie bei den empfindungsfähigen Maschinen in Spielfilmen). Viele Anwendungen dieser Technologie lösen die verschiedenartigsten Aufgaben. Dieses Buch soll sich auf die Grundlagen des Deep Learnings konzentrieren, das sowohl in der topaktuellen Forschung als auch in der Industrie zum Einsatz kommt, und dir diese Kenntnisse vermitteln.

## 2.2 Was ist Machine Learning?

*Ein Fachgebiet, das Computern die Fähigkeit verleiht, zu lernen, ohne explizit programmiert zu werden.*

– Arthur Samuel zugeschrieben

Wenn also Deep Learning eine Teilmenge des Machine Learnings ist, was ist dann eigentlich Machine Learning? Ganz allgemein ist es das, was der Name besagt. Machine Learning ist ein Teilgebiet der Informatik, das sich damit befasst, dass *Maschinen lernen*, Aufgaben zu erledigen, für die sie *nicht explizit programmiert* wurden. Oder kurz und bündig: Maschinen beobachten ein Muster und versuchen, es irgendwie zu imitieren, entweder direkt oder indirekt:

Machine Learning ~ = Affe sieht, Affe tut

Ich erwähne hier direktes und indirektes Imitieren, um die Analogie zu den beiden Haupttypen des Machine Learnings aufzuzeigen: *überwachtes* und *unüberwachtes* Lernen. Überwachtes Machine Learning ist das direkte Imitieren des Musters, das einem Datensatz zu eigen ist, bei einem anderen Datensatz. Es ist der Versuch, eine Eingabedatenmenge in eine Ausgabedatenmenge umzuwandeln. Das kann eine unglaublich mächtige und nützliche Fähigkeit sein. Sieh dir die folgenden Beispiele an (die Eingabedatenmengen sind fett gedruckt, die Ausgabedatenmengen kursiv):



- Verwendung der **Pixel** eines Bilds, um das *Vorhandensein* oder das *Fehlen einer Katze* zu erkennen
- Verwendung der **Filme, die dir gefallen haben**, um *Filme* vorherzusagen, *die dir wahrscheinlich gefallen werden*
- Verwendung der **Wörter**, die jemand benutzt, um festzustellen, ob die Person *fröhlich* oder *traurig* ist
- Verwendung der **Daten einer Wetterstation**, um die *Regenwahrscheinlichkeit* vorherzusagen
- Verwendung von **Motorsensoren**, um die optimalen *Einstellungen* zu ermitteln
- Verwendung von **Nachrichtmeldungen**, um die morgigen *Aktienkurse* vorherzusagen
- Verwendung eines **Eingabewerts** zur Berechnung des *doppelten Werts*
- Verwendung der Rohdaten einer **Audiodatei**, um eine *Transkription* des Inhalts zu erstellen

All diese Aufgaben gehören zum überwachten Lernen. Der Machine-Learning-Algorithmus versucht stets, das Muster, das die beiden Datenmengen verbindet, so zu imitieren, dass *die eine Datenmenge zur Vorhersage der anderen* verwendet werden kann. Stell dir vor, du könntest bei all diesen Beispielen die *Ausgabedatenmenge* nur anhand der *Eingabedatenmenge* vorhersagen – das wäre schon eine beachtliche Leistung.

## 2.3 Überwachtes Machine Learning

### Überwachtes Lernen transformiert Datenmengen.

Überwachtes Lernen ist ein Verfahren, um eine Datenmenge in eine andere umzuwandeln. Wenn du beispielsweise eine Datenmenge namens *Aktienkurse am Montag* hättest, in der die Kurse aller Aktien an den Montagen der letzten zehn Jahre aufgezeichnet sind, und eine zweite, die die Aktienkurse an den Dienstagen desselben Zeitraums enthält, könnte ein überwachter Lernalgorithmus die eine Datenmenge verwenden, um die andere vorherzusagen.



Wenn du einen überwachten Machine-Learning-Algorithmus erfolgreich mit den Daten von zehn Jahren trainiert hast, bist du in der Lage, den Aktienkurs für einen

beliebigen Dienstag in der Zukunft vorherzusagen, wenn dir der Aktienkurs am direkt vorhergehenden Montag bekannt ist. Lege eine kurze Pause ein und denke einen Moment darüber nach.

Überwachtes Machine Learning gehört bei der angewandten künstlichen Intelligenz (der sogenannten schwachen KI) zum Alltag. Es erweist sich als nützlich, um das, *was du weißt*, als Eingabe zu verwenden und schnell in das umzuwandeln, *was du wissen möchtest*. Auf diese Weise können Machine-Learning-Algorithmen die Intelligenz und die Fähigkeiten des Menschen auf unzählige Arten erweitern.

Bei der Nutzung der Ergebnisse von Machine Learning besteht der größte Aufwand darin, einen überwachten Klassifikator zu trainieren. Bei der Entwicklung möglichst genauer überwachter Machine-Learning-Algorithmen kommt typischerweise sogar unüberwachtes Machine Learning (dazu gleich mehr) zum Einsatz.



Im folgenden Teil des Buchs wirst du Algorithmen entwickeln, die Eingabedaten mit bestimmten Eigenschaften entgegennehmen. Sie sind beobachtbar, lassen sich aufzeichnen, und sie sind *nachvollziehbar*. Außerdem lassen sie sich in wertvolle Ausgabedaten umwandeln, für die eine logische Analyse erforderlich ist. All das leistet überwachtes Machine Learning.

## 2.4 Unüberwachtes Machine Learning

### Unüberwachtes Lernen gruppiert deine Daten.

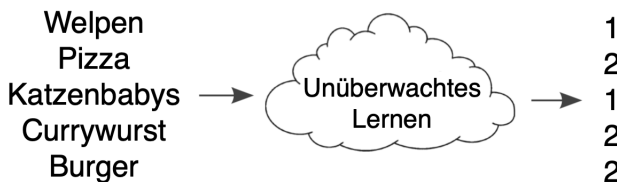
Unüberwachtes und überwachtes Lernen haben etwas gemeinsam: Eine Datenmenge wird in eine andere Datenmenge umgewandelt. Beim unüberwachten Lernen ist die zu transformierende Datenmenge allerdings vorher *nicht bekannt*. Im Gegensatz zum überwachten Lernen gibt es keine »richtige Antwort«, die das Modell reproduzieren soll. Ein unüberwachter Lernalgorithmus sucht einfach nur nach Mustern in den Daten und zeigt an, was er gefunden hat.

Das Aufteilen einer Datenmenge in Gruppen, das sogenannte *Clustering*, ist eine Form des unüberwachten Lernens. Beim Clustering wird eine Menge von Datenpunkten in eine Sequenz von Clusterbezeichnungen transformiert. Wenn eine Aufteilung in zehn Cluster vorgenommen wird, verwendet man für die Bezeichnungen üblicherweise die Zahlen von 1 bis 10. Jedem Datenpunkt wird eine Zahl zugewiesen, die angibt, zu welchem Cluster er gehört. Die Menge aus Datenpunk-

ten wird also in eine Menge von Bezeichnungen umgewandelt. Und warum sind die Bezeichnungen Zahlen? Der Algorithmus sagt nichts darüber aus, was die Cluster eigentlich sind. Er stellt lediglich fest: »Hey, Forscher! Ich habe eine Struktur entdeckt. Sieht ganz danach aus, als ob es Gruppen in den Daten gibt – und zwar diese hier!«



Eine gute Nachricht: Dieses Prinzip des Clusterings kannst du dir als Definition des unüberwachten Lernens merken. Es gibt zwar eine Vielzahl verschiedener Arten, aber *alle Formen des unüberwachten Lernens können als eine Form des Clusterings betrachtet werden*. Mehr zu diesem Thema später im Buch.

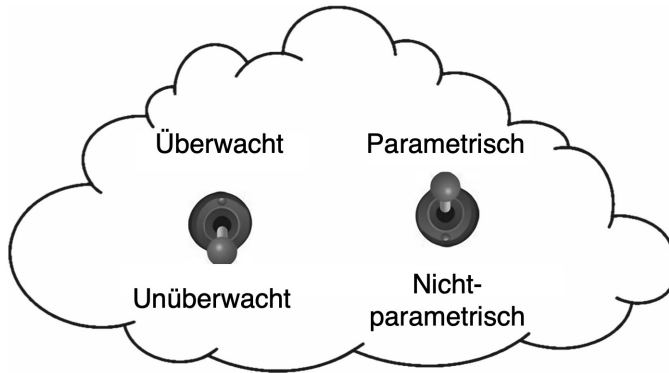


Sieh dir das Beispiel an. Der Algorithmus gibt zwar nicht an, wie die Cluster heißen, aber du kannst sicher erkennen, wonach er die Wörter gruppiert hat. (Antwort: 1 == niedlich, 2 == lecker.) Später werden wir uns damit befassen, dass andere Formen des unüberwachten Lernens auch nur eine Art Clustering sind und weshalb sich diese Cluster für das überwachte Lernen als nützlich erweisen.

## 2.5 Parametrisches und nichtparametrisches Lernen

### Grob vereinfacht: Lernen durch Trial and Error vs. Zählen und Wahrscheinlichkeit

Auf den letzten Seiten wurden alle Machine-Learning-Algorithmen einer von zwei Gruppen zugeordnet: überwachtem und unüberwachtem Lernen. Jetzt werden wir eine andere Variante erörtern, die gleichen Machine-Learning-Algorithmen in zwei Gruppen zu unterteilen, nämlich in parametrische und nichtparametrische. Wenn wir unsere kleine Machine-Learning-Wolke genau betrachten, stellen wir fest, dass sie über zwei Einstellungen verfügt:



Wie du siehst, gibt es eigentlich vier verschiedene Arten von Algorithmen. Ein Algorithmus ist entweder unüberwacht oder überwacht und entweder parametrisch oder nichtparametrisch. Im vorangegangenen Abschnitt über überwachtes Lernen ging es um *die Art des erlernten Musters*, bei der Parametrisierung hingegen geht es darum, wie das Gelernte *gespeichert* wird, und im weiteren Sinne oft um die *Lernmethode*. Sehen wir uns zunächst einmal die formale Definition von parametrischen und nichtparametrischen Algorithmen an. Man hat sich übrigens noch nicht auf einen genauen Unterschied einigen können.

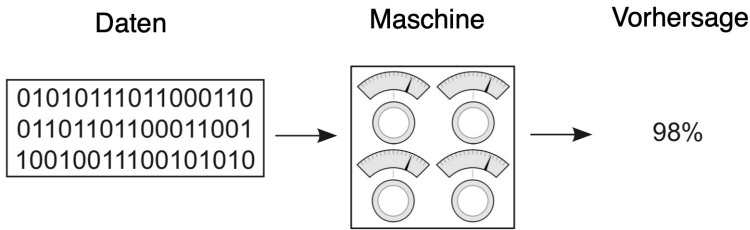
*Ein parametrisches Modell ist dadurch gekennzeichnet, dass es eine feste Anzahl von Parametern besitzt, die Anzahl der Parameter eines nichtparametrischen Modells hingegen ist unbegrenzt (sie wird durch die Daten bestimmt).*

Betrachten wir als Beispiel die Aufgabe, einen quadratischen Zapfen in eine passende (quadratische) Öffnung zu stecken. Manche Menschen (etwa Kleinkinder) stopfen den Zapfen einfach in alle Öffnungen, bis er irgendwo passt (parametrisch). Ein Teenager hingegen würde vielleicht die Anzahl der Seiten (vier) zählen und nach einer Öffnung suchen, die ebenso viele Seiten besitzt (nichtparametrisch). Parametrische Modelle verwenden tendenziell das Prinzip von Trial and Error, nichtparametrische dagegen versuchen eher, zu zählen. Das sehen wir uns genauer an.

## 2.6 Überwachtes parametrisches Lernen

### Grob vereinfacht: Lernen durch Trial and Error mithilfe von Drehschaltern

Überwachte parametrisch lernende Maschinen besitzen eine festgelegte Anzahl von Drehschaltern (das ist der parametrische Teil), wobei das Lernen durch das Drehen der Schalter erfolgt. Vorliegende Eingabedaten werden entsprechend der Winkel, in dem die Drehschalter stehen, verarbeitet und in eine *Vorhersage* umgewandelt.



Das Lernen vollzieht sich in der Form, dass die Drehschalter in verschiedenen Winkeln positioniert werden. Wenn du versuchst, vorherzusagen, dass Bayern München die Champions League gewinnt, dann würde dieses Modell zuerst Daten entgegennehmen (wie Statistiken über Siege/Niederlagen oder die durchschnittliche Anzahl der Zehen pro Spieler) und eine Vorhersage treffen (etwa eine Chance von 98 Prozent). Anschließend würde das Modell beobachten, ob die Bayern tatsächlich gewinnen. Wenn feststeht, dass sie gewonnen haben, würde der Lernalgorithmus die *Position der Drehschalter aktualisieren*, um beim nächsten Mal eine genauere Vorhersage zu treffen, wenn die *gleichen oder ähnliche Eingabedaten* vorliegen.

Wenn sich der Schalter für »Siege/Niederlagen« bei der Vorhersage als nützlich erwiesen hat, würde das Modell ihn vielleicht etwas »aufdrehen«. Umgekehrt würde es den Schalter für die »durchschnittliche Anzahl der Zehen pro Spieler« »zurückdrehen«, wenn dieser Datenpunkt nicht besonders hilfreich war. Auf diese Weise lernen parametrische Modelle!

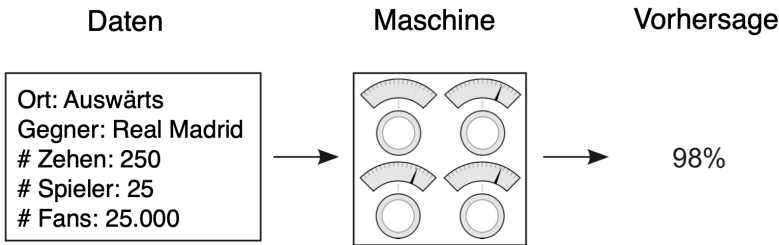
Hier ist zu beachten, dass alles, was das Modell gelernt hat, zu jedem Zeitpunkt in Form der Positionen der Drehschalter erfasst werden kann. Du kannst dir diese Art Lernmodell auch wie einen Suchalgorithmus vorstellen. Du »suchst« nach den geeigneten Stellungen der Drehschalter, indem du Konfigurationen ausprobierst, anpasst und wieder testest.

Außerdem musst du bedenken, dass das Prinzip von Trial and Error nicht der formalen Definition entspricht, es wird in parametrischen Modellen aber (mit Ausnahmen) häufig verwendet. Wenn es eine beliebige (aber feststehende) Anzahl von Drehschaltern gibt, muss eine Suche durchgeführt werden, um die optimale Konfiguration zu finden. Das ist beim nichtparametrischen Lernen anders, das oft auf Zählvorgängen beruht und (hin und wieder) neue Drehschalter hinzufügt, wenn etwas Neues gefunden wird, das gezählt werden kann. Wir unterteilen überwachtes parametrisches Lernen in drei Schritte.

### Schritt 1: Vorhersage treffen

Zur Veranschaulichung des überwachten parametrischen Lernens bleiben wir bei der Analogie, vorherzusagen, ob Bayern München die Champions League gewinnt. Der erste Schritt besteht wie erwähnt darin, Statistiken zusammenzutragen, sie in

die Maschine einzuspeisen und eine Vorhersage über die Wahrscheinlichkeit zu treffen, dass die Bayern gewinnen.



### Schritt 2: Vergleich mit Faktum

Der zweite Schritt besteht darin, die Vorhersage (98 Prozent) mit dem Faktum zu vergleichen, das von Bedeutung ist (ob Bayern München gewonnen hat). Leider haben sie verloren, somit ergibt sich:

Vorhersage: 98 % > Faktum: 0 %

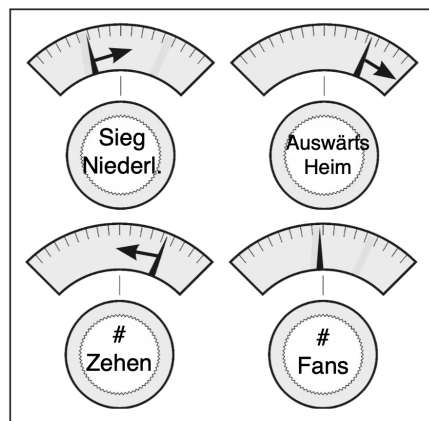
Dieser Schritt erfasst, dass das Modell die bevorstehende Niederlage genau vorhergesehen hätte, wenn die Vorhersage 0 Prozent Gewinnchance gelaundet hätte. Die Maschine soll möglichst genau arbeiten, was zu Schritt 3 führt.

### Schritt 3: Erlernen des Musters

Dieser Schritt passt die Position der Drehschalter an und berücksichtigt dabei, wie sehr die Vorhersage daneben gelegen hat (98 Prozent) und was die Eingabedaten zum Zeitpunkt der Vorhersage waren (die Statistiken über Siege/Niederlagen). Auf diese Weise soll anhand der Eingabedaten eine genauere Vorhersage erfolgen.

Theoretisch sollte dieser Schritt beim nächsten Vorliegen der gleichen Statistiken über Siege/Niederlagen weniger als 98 Prozent vorhersagen. Beachte auch, dass jeder Drehschalter die *Empfindlichkeit der Vorhersage für verschiedene Typen von Eingabedaten* repräsentiert. Das ist das, was du änderst, wenn die Maschine »lernt«.

Empfindlichkeit anpassen durch Drehen der Schalter

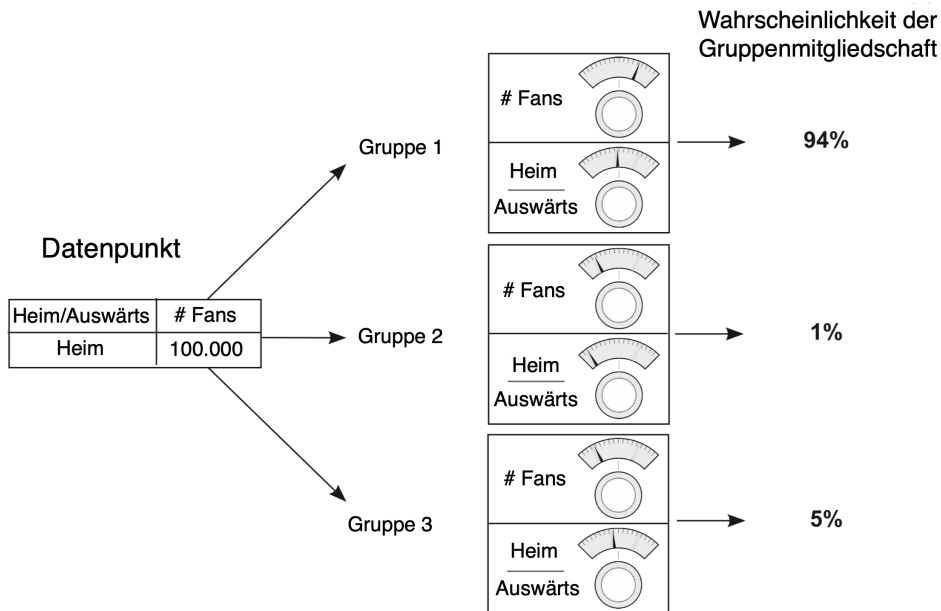


## 2.7 Unüberwachtes parametrisches Lernen

Heim/Auswärts	# Fans
<b>Heim</b>	<b>100.000</b>
Auswärts	50.000
<b>Heim</b>	<b>100.000</b>
<b>Heim</b>	<b>99.000</b>
Auswärts	50.000
Auswärts	10.000
Auswärts	11.000

Das unüberwachte parametrische Lernen verwendet einen sehr ähnlichen Ansatz. Wir gehen die Schritte einmal der Reihe nach durch. Wie du weißt, geht es beim unüberwachten Lernen immer um das Gruppieren von Daten. Das unüberwachte *parametrische* Lernen verwendet die Drehschalter, um Daten zu gruppieren. Aber in diesem Fall gibt es für jede Gruppe mehrere Drehschalter, die jeweils die Stärke der Zugehörigkeit der Eingabedaten zu einer bestimmten Gruppe angeben (mit Ausnahmen – das ist eine allgemeine Beschreibung). Betrachten wir ein Beispiel, bei dem du die Daten in drei Gruppen unterteilen möchtest.

Die Datenmenge enthält drei Cluster, die das parametrische Modell entdecken soll. Sie sind durch Textformatierung als **Gruppe 1**, Gruppe 2 und Gruppe 3 gekennzeichnet. Nun übergeben wir einem trainierten unüberwachten Modell den ersten Datenpunkt. Die Zuordnung (auch *Mapping* genannt) zur **Gruppe 1** ist am stärksten.



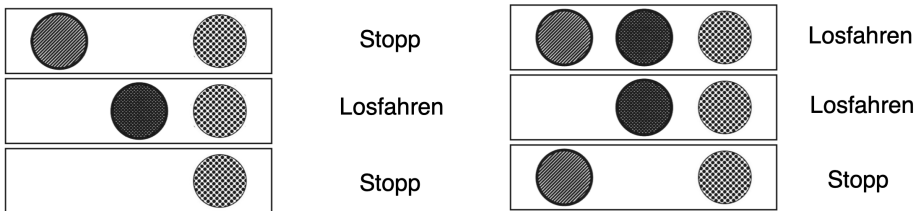
Die Maschine jeder Gruppe versucht, die Eingabedaten in eine Zahl zwischen 0 und 1 umzuwandeln und somit die Wahrscheinlichkeit anzugeben, dass die Eingabe

bedaten zu dieser Gruppe gehören. Das Training für die Modelle unterscheidet sich deutlich und führt zu unterschiedlichen Eigenschaften, aber ganz allgemein werden die Parameter angepasst, um die Eingabedaten auf die zugehörige/n Gruppe/n abzubilden.

## 2.8 Nichtparametrisches Lernen

### Grob vereinfacht: Auf Zählvorgängen beruhende Verfahren

Nichtparametrisches Lernen ist eine Algorithmenklasse, bei der die Anzahl der Parameter von den Daten abhängt (und nicht vorab festgelegt ist). Daher werden im Allgemeinen Verfahren verwendet, die auf die eine oder andere Weise zählen und so die Anzahl der Parameter anhand der in den Daten vorhandenen Objekte erhöhen. Bei einem überwachten nichtparametrischen Modell könnte beispielsweise gezählt werden, wie oft das Auftreten einer bestimmten Farbe einer speziellen Ampel dafür sorgt, dass Autos »losfahren«. Schon nach dem Zählen einiger weniger Beispiele wäre das Modell in der Lage vorherzusagen, dass das Aufleuchten der *mittleren* Lampe immer (in 100 Prozent der Fälle) dafür sorgt, dass Autos losfahren und das Aufleuchten der *rechten* Lampe nur manchmal (in 50 Prozent der Fälle).



Dieses Modell hätte drei Parameter: drei Zähler, die angeben, wie oft eine der Lampen aufleuchtet und Autos losfahren (eventuell geteilt durch die Anzahl der Beobachtungen). Bei fünf Lampen gäbe es fünf Zähler (fünf Parameter). Dieses einfache Modell ist *nichtparametrisch*, weil es die Eigenschaft hat, dass die Anzahl der Parameter sich in Abhängigkeit von den Daten (hier die Anzahl der Lampen) ändert. Das ist bei parametrischen Modellen anders, bei denen die Anzahl der Parameter anfangs zwar festgelegt ist, diese aber nach Ermessen des Forschers, der das Modell trainiert, erhöht oder verringert werden kann (unabhängig von den Daten).

Ein kritischer Beobachter könnte das infrage stellen. Beim vorangegangenen parametrischen Modell gab es offenbar für jeden Eingabedatenpunkt einen Drehschalter. Die meisten parametrischen Modelle benötigen irgendeine Art *Eingabe*, die von der Anzahl der Klassen in den Daten abhängt. Es gibt also eine *Grauzone* zwischen parametrischen und nichtparametrischen Algorithmen. Auch parametrische Algo-



rithmen werden in gewisser Weise von der Anzahl der Klassen in den Daten beeinflusst, sogar wenn es sich überhaupt nicht um ein Zählmuster handelt.

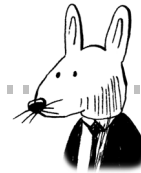
Hier wird auch deutlich, dass *Parameter* ein allgemeiner Begriff ist, der sich lediglich auf die Menge der Zahlen bezieht, die zum Modellieren eines Musters verwendet werden (ohne Berücksichtigung, wie diese Zahlen verwendet werden). Zähler sind Parameter. Gewichte sind Parameter. Normalisierte Varianten von Zählern und Gewichten sind ebenfalls Parameter. Korrelationskoeffizienten können auch Parameter sein. Der Begriff bezeichnet die Menge der Zahlen, die zur Modellierung eines Musters verwendet werden. Und Deep Learning ist eine Klasse von parametrischen Modellen. Ich werde in diesem Buch nicht näher auf nichtparametrische Modelle eingehen, aber es handelt sich um eine hochinteressante und leistungsstarke Algorithmenklasse.

## 2.9 Zusammenfassung

In diesem Kapitel haben wir die verschiedenen Ausprägungen des Machine Learnings etwas genauer betrachtet. Du hast erfahren, dass ein Machine-Learning-Algorithmus entweder überwacht oder unüberwacht und entweder parametrisch oder nichtparametrisch ist. Darüber hinaus haben wir erkundet, was genau diese vier Gruppen von Algorithmen unterscheidet. Du hast gelernt, dass überwachtes Machine Learning zu einer Algorithmenklasse gehört, bei der eine Datenmenge anhand einer anderen vorhergesagt wird, und dass unüberwachtes Lernen im Allgemeinen eine einzelne Datenmenge zu verschiedenen Clustern gruppiert. Zudem hast du erfahren, dass parametrische Algorithmen eine festgelegte Anzahl von Parametern besitzen und dass nichtparametrische Algorithmen die Anzahl der Parameter den Daten entsprechend anpassen.

Deep Learning verwendet sowohl für überwachte als auch für unüberwachte Vorhersagen neuronale Netze. Bislang sind wir auf theoretischer Ebene verblieben, damit du die Bedeutung für das Fachgebiet als Ganzes erkennst. Im nächsten Kapitel wirst du dein erstes neuronales Netz entwickeln, und alle nachfolgenden Kapitel sind *projektbasiert*. Also starte dein Jupyter Notebook und los geht's!

# Stichwortverzeichnis



\*-Operator 61

## A

Abbruch

früher 176

Ableitung 85, 86, 88, 201

Absolutwert 69

Abstimmung des Sprachmodells 327

Aggregation

sichere 338

Ähnlichkeit 194

Aktivierung 64

Aktivierungsfunktion 187

Anleitung 197

Bedingungen 188

Aktualisierung der Gewichte 263

alpha 94

Ampelproblem 121

Architektur 163, 228

Ausgabe

mehrere 52, 54

Autograd 274

erweitern 276

Indizierung hinzufügen 297

Automatische Optimierung 288

## B

Babi-Datensatz 258

Backpropagation 142, 159, 262

Minibatch 312

RNN 318

Truncated 312

Bag-of-words 250

Batch-Gradientenabstieg 132, 183

Bedingte Korrelation 146, 189

Berechnungsgraph 273

dynamischer 274

Binäre Wahrscheinlichkeit 192

## C

Chiffre 338

Clustering 28

## D

Datenmatrix 125

Datenvorverarbeitung 123

Deaktivieren von Knoten 147

Deep Learning

Auswirkungen 20

Definition 25

Programmiersprache 24

delta 77

Differenzielle Privatsphäre 337

Dimensionalität 162

Divergenz 94

dot-Funktion 63

Dropout-Verfahren 177, 179

Bewertung 182

Druck auf Gewichte 133

## E

Eingabe 41

mehrere 44, 54

Einheitsmatrix 250

Elementweise Operation 47

Embedding  
   Reihenfolge 249  
 Embedding-Schicht 296  
 Enron-Datensatz 332  
 Erlernen 66  
 Erlernen des Musters 32  
 Euklidischer Abstand 230  
 Euler'sche Zahl 196  
 Explodierender Gradient 317  
 Exponenzierung 196

**F**

Faltung 206  
 Faltungskern 207  
 Faltungsschicht 206  
 Federated Learning 330  
 Fehler 66  
   messen 68  
   mittlerer quadratischer 66  
   reiner 74  
 Fehlerattribution 66, 134  
 Fehlerfläche 111  
 Fehlerfunktion 238  
 Fehlermaß 133  
 Form 38, 62  
 Forward Propagation 64, 148, 261  
 Framework 269  
   erlernen 293  
 Früher Abbruch 176  
 Funktion  
   sigmoide 191

**G**

Gate 320  
 Gemittelter Wortvektor 246  
 Gewicht 39  
   Beitrag zum Fehler 150  
   einfrieren 109  
   negatives 48  
   Visualisierung 118  
 Gewichtete Summe 45, 46, 56  
 Glaubhafte Abstreitbarkeit 337  
 Gradient  
   explodierender 317  
   verschwindender 317  
 Gradientenabstieg 74, 90  
   Batch- 132  
   Eigenschaften 102  
   mehrere Ein- und Ausgaben 114  
   mit mehreren Eingaben 99  
   stochastischer 131  
   vollständiger 132

**H**

Homomorphe Verschlüsselung 338  
 Hot und Cold Learning 69

**I**

IMDB-Filmbewertungsdatenbank 220  
 Indirekte Korrelation 139  
 Infinitesimalrechnung 88

**J**

Jupyter Notebook 23

**K**

Kapazität 178  
 Klartext 338  
 Klassifikation  
   von Texten 220  
 Knoten  
   deaktivieren 147  
 Kontrastveränderung 196  
 Konvexe Optimierung 189  
 Korrekturklassifikationsrate 172  
 Korrelation 118, 123, 133  
   bedingte 146, 189  
   Ein- und Ausgabeschicht 139  
   erzeugen 140  
   indirekte 139  
 Korrelationszusammenfassung 158  
 Kreuzentropie 202  
 Kreuzentropie-Schicht 300  
 Krümmung 111, 190

**L**

Lautstärkeregler 43  
 LEGO-Stein 161  
 Lernen  
   nichtparametrisches 29, 34  
   parametrisches 29  
   überwachtes 26  
   unüberwachtes 28  
 Lineare Schicht 224, 290  
 Logisches AND 49  
 Logisches NOT 49  
 LSTM-Schicht 322  
 LSTM-Sprachmodell 325  
 LSTM-Zelle 320  
 Lücke  
   ausfüllen 233

**M**

Machine Learning  
   Definition 26

- überwachtes 27
- unüberwachtes 28
- Matrix 57, 125
  - in Python 128
  - transponierte 285
- Max-Pooling 208
- Mean-Pooling 208
- Mehrere Ausgaben 52, 54
- Mehrere Ein- und Ausgaben 54
- Mehrere Eingaben 44, 54
- Minimum einer Funktion 89
- Mittelwert der Aktualisierung 184
- Mittlerer quadratischer Fehler 66
- MNIST-Datensatz 116
- MNIST-Ziffernklassifizierung 170, 193, 201
- Multiplikation 41

**N**

- Natural Language Processing (NLP) 218
- Negation 280
- Negative Umkehrung 75
- Negatives Gewicht 48
- Negatives Sampling 233
- Netz
  - neuronales 39
- Neuron 158, 232
- Neuronales Netz 39
  - Architektur 228
  - Kapazität 178
  - Überanpassung 174
  - Visualisierung 160
- Nichtlineare Schicht 294
- Nichtlinearität 147
- Nichtparametrisches Lernen 29, 34
- np.dot-Funktion 186
- NumPy 23, 51, 59, 60

**O**

- One-hot-Codierung 222
- Operation
  - elementweise 47
- Optimierung
  - automatische 288
  - konvexe 189

**P**

- Parametrisches Lernen 29
- Perplexität 264
- Privatsphäre 329
  - differenzielle 337
- Python Codecademy 24

**R**

- Randomized-Response-Technik 337
- Rauschen 175
- Regularisierung 138, 176
- Reiner Fehler 74
- Rekurrente Neuronale Netze 250
- Rekurrente Schicht 302
- relu-Funktion 187
- Repräsentation
  - verlustfreie 128

**S**

- Sampling
  - negatives 233
- Satzvektor
  - vergleichen 244
- Schicht
  - Kreuzentropie- 300
  - lineare 224, 290
  - nichtlineare 294
  - rekurrente 302
  - verdeckte 228
- Schichttyp 290
- Sensitivität 84
- Sichere Aggregation 338
- Sigmoide Funktion 191
- Signal 176
- Skalarprodukt 46, 48, 56, 119
- Skalierung 75
- softmax 193, 195, 202
- softmax-Funktion 260
- Spam-Erkennung 332
- Sprachmodellierung 309
- Sprachverarbeitung
  - überwachte 219
- Steigung 86
- Stetigkeit 188
- Stochastischer Gradientenabstieg 131
- Stoppen 75
- Struktur 206
- Suchalgorithmus 31
- Summe
  - gewichtete 45, 46, 56
- Sum-Pooling 208

**T**

- tanh-Funktion 191
- Temperaturvorhersage 192
- Tensor
  - Definition 271
  - mehrfach verwendeter 275

Tensor-Klasse 279  
Tensoroperation 281  
Test-Korrektklassifikationsrate 172  
Transponierte Matrix 285  
Trial and Error 30  
Truncated Backpropagation 312

## U

Überanpassung 135, 170, 205  
    Rauschen 179  
Übergangsmatrix 253, 255  
Überwachte Sprachverarbeitung 219  
Überwachtes Lernen 26  
Umkehrung  
    negative 75  
Unüberwachtes Lernen 28

## V

Validierungsdatenmenge 177  
Vektor 47  
Vektor-Matrix-Multiplikation 57, 164  
Vektorrechnung 47  
Verarbeitung natürlicher Sprache 218  
Verdeckte Schicht 228  
Vergleich 66  
Vergleich mit Faktum 32  
Verlagerung von Arbeitsplätzen 20

Verlustfreie Repräsentation 128  
Verlustfunktion 237, 238  
Verlustfunktions-Schicht 292  
Verschlüsselung  
    homomorphe 338  
Verschwindender Gradient 317  
Visualisierung mit Buchstaben 165  
Vokabular 260  
Vorhersage 37, 41  
Vorhersage treffen 31

## W

Wahrscheinlichkeit  
    binäre 192  
Wortähnlichkeit 248  
Wortanalogie 239, 240  
Wort-Embedding 239  
    vergleichen 230  
Wortkorrelationen 222  
Wortvektor  
    Mittelwertbildung 246

## Z

Zelle 305  
Zielfunktion 238  
Ziffernklassifizierung 170  
Zufälligkeit 133