Verfahren für welche Aufgaben geeignet sind, und Sie können die meisten dieser Verfahren selbst implementieren. Sie können eigene neuronale Netze aufbauen und anlernen, um Texte zu klassifizieren, zu generieren oder in eine andere Sprache zu übersetzen.

Darüber hinaus bietet Ihnen dieses Buch einen Einstieg in die Verwendung neuronaler Netze, die von anderen vortrainiert und zur Verfügung gestellt werden. Sie lernen, wie Sie allgemeine Sprachmodelle, die über sehr große Datenmengen angelernt wurden, laden und einsetzen können, und Sie lernen, wie Sie diese Netze nachtrainieren können, um sie für spezifische Aufgaben nutzbar zu machen.

Welche Inhalte Sie erwarten

Dieses Buch bietet eine schrittweise und kompakte Einführung in den Themenkomplex Machine Learning/Natural Language Processing (NLP). Die Kapitel bauen inhaltlich aufeinander auf. Wenn Sie also eines der Kapitel lesen, sollten Sie die Inhalte aus den vorherigen Kapiteln kennen. Konzepte, Klassen oder Funktionen werden in der Regel nur an einer Stelle erklärt, und Sie können davon ausgehen, dass einmal eingeführtes Wissen später wieder gebraucht wird. Wenn Sie neu in diesem Gebiet sind, ist es daher ratsam, die Kapitel nacheinander zu lesen, damit das didaktische Konzept aufgeht. Wenn Sie schon über Kenntnisse im einen oder anderen Bereich verfügen, können Sie natürlich einzelne Kapitel oder auch mehrere Kapitel überspringen.

Wir starten unsere Reise in die Welt des Deep Learnings mit einer allgemeineren Einführung in den Bereich maschinellen Lernens. Bevor wir uns an komplexen Modellen abarbeiten, sollten wir eine konkrete Vorstellung davon haben, wie einfache Modelle funktionieren. Wir machen uns also mit den Grundlagen statistischer Lernverfahren und den Basics im Umgang mit Daten in Python vertraut.

Deshalb beginnen wir im Anschluss an die Einleitung im **zweiten Kapitel** mit einer Einführung in die Behälterklassen, die Python standardmäßig bietet, und in einige externe Pakete und Funktionen wie *NumPy*, die uns die Arbeit mit Textdaten und Matrizen erleichtern. Das **dritte Kapitel** widmet sich den allgemeinen Grundlagen maschinellen Lernens. Wir schauen uns an, wie einfache Lernzellen arbeiten, wie man Beziehungen zwischen *x*- und *y*-Variablen modelliert, was es mit der Optimierung von Gewichten und dem Gradientenabstiegsverfahren auf sich hat und wie sich mithilfe von Aktivierungsfunktionen beliebige Zielwerte (stetige und kategoriale Werte) produzieren lassen.

In **Kapitel 4** steigen wir dann in die Analyse von Textdaten ein. Dabei geht es zunächst um bewährte Vorverarbeitungsverfahren wie den Bag-of-Words-Ansatz, der es

ermöglicht, Textdaten auf sehr basale Art mit maschinellen Standardverfahren zu verarbeiten. **Kapitel 5** bietet im Anschluss einen Einstieg in das eigentliche Zielthema des Buches: in die Arbeit mit neuronalen Netzen. Auch hier müssen wir uns zuerst einiger grundlegender Verfahren versichern, ehe wir uns um die speziellen, auf die Interpretation und Produktion von Texten zugeschnittenen Architekturen kümmern. In diesem Kapitel geht es daher um die Basics, die für die praktische Arbeit mit neuronalen Netzen erforderlich sind. Angefangen von der Funktionsweise einzelner Neuronen, über Schichten von Neuronen, das Durchschleusen von Daten durch diese Schichten bis hin zur Verlustfunktion und der Optimierung der Gewichte mithilfe des Backpropagation-Mechanismus. Danach wissen Sie, wie ein neuronales Netz lernt, wie Gewichte initialisiert und optimiert werden und welche Aufgaben Ihnen bei der Zusammenstellung eines solchen Netzes zukommen.

Mit diesem Wissen ausgerüstet, erörtern wir in **Kapitel 6** eine erste, für die Verarbeitung von Textdaten konzipierte Netzarchitektur: rekurrente Netze. Dabei handelt es sich um ein Verfahren, das im Speziellen die zeitliche Abfolge der Präsentation von Zeichen – zum Beispiel von Buchstaben in Wörtern oder von Wörtern in Sätzen – berücksichtigt und auf dieser Grundlage Interpretationen erzeugt. In **Kapitel 7** geht es nahtlos mit konvolutionalen Netzen weiter. Zwar hat sich dieses Verfahren vor allem bei der Verarbeitung von Bilddaten bewährt, es lässt sich aber ohne Weiteres auf Sequenzdaten übertragen. Die Vor- und Nachteile gegenüber rekurrenten Netzen sehen wir uns dabei anhand einer automatischen Rechtschreibkorrektur genauer an. Das folgende **Kapitel 8** thematisiert dann die Vorverarbeitung von Wörtern mit Worteinbettungsverfahren. Damit lassen sich Wörter bzw. Token als Vektoren in einem mehrdimensionalen Raum unter Berücksichtigung semantischer und grammatikalischer Beziehungen darstellen. Das Verfahren bildet heute die Grundlage für fast alle Deep Learning-Ansätze, die Textdaten als Rohmaterial verwenden. Wir beleuchten dabei nicht nur die Möglichkeiten, Worteinbettungen selbst anzulernen, sondern führen auch in die Verwendung vortrainierter Vektorräume wie *fastText* oder *Word2Vec* ein.

Beginnend mit **Kapitel 9** eruieren wir dann die Optionen, die komplexe neuronale Netze bei der Textinterpretation und Textgenerierung bieten. Insbesondere geht es um Encoder-Decoder-Modelle. Sie sind aus der modernen Textanalyse (sei es in Sequence-to-Sequence- oder in Transformer-Modellen) nicht mehr wegzudenken. Zunächst dreht sich aber alles um das Handwerkszeug, das wir brauchen, um solche Modelle praktisch umzusetzen: Es geht darum, wie wir Netze, die über zwei Eingänge verfügen oder in denen Datenströme bestimmte Schichten überspringen, aufsetzen können. Diese Kenntnisse sind notwendig, wenn wir im nachfolgenden **Kapitel 10** Sequence-to-Sequence-Modelle genauer unter die Lupe nehmen. Mit dieser Architektur lassen sich

nicht nur einzelne Wörter, sondern ganze Sätze oder Textpassagen erzeugen. Solche Architekturen werden für Übersetzungen oder Textzusammenfassungen eingesetzt. Darüber hinaus lernen wir in diesem Kontext den Attention-Mechanismus kennen, der in einer Variante, der Self-Attention, auch in Transformer-Modellen Karriere gemacht hat. Das ist das Thema von **Kapitel 11**. Darin zeigen wir nicht nur, wie Transformer-Modelle funktionieren, sondern wir sehen uns auch an, wie sich vortrainierte Modelle, die über den Transformer-Hub *Hugging Face* vertrieben werden, sich für verschiedene Aufgaben nachtrainieren lassen. Das Buch schließt im **zwölften Kapitel** mit einer Zusammenfassung einiger grundlegender Konzepte und einer Diskussion der Stärken und Schwächen neuronaler Netze.

Wie Sie mit den Codebeispielen arbeiten können

Die meisten Kapitel bestehen aus einem Mix aus Einführungen in Konzepte maschinellen Lernens und aus einem praxisorientierten Teil, in denen diese Konzepte mit Python umgesetzt werden. Wenn Sie die Beispiele auf Ihrem Rechner laufen lassen möchten, müssen Sie die Voraussetzungen dafür schaffen.

Wir verwenden Python in der *Version 3.7.6*, eingebettet in eine virtuelle Umgebung, die über die Data Science-Plattform *Anaconda* verwaltet wird. Anaconda bietet unter anderem den Vorteil, dass es bei der Installation externer Pakete sicherstellt, dass die Pakete untereinander harmonieren. Da wir einige Pakete installieren müssen und da diese Pakete in ihrer Arbeit wiederum auf weitere Unterpakete angewiesen sind, ist das ziemlich hilfreich.

Um also arbeitsfähig zu sein, benötigen Sie die folgenden Bibliotheken in Ihrer virtuellen Umgebung:

```
matplotlib, Version=3.4.2
nltk, Version=3.6.2
numpy, Version=1.19.2
pandas, Version=1.0.3
scikit-learn, Version=0.22.1
tensorflow, Version=2.3.0
transformers, Version=4.10.2
```

Bis auf *Transformers* (das zum Zeitpunkt der Drucklegung mit pip installiert werden muss) sind alle Bibliotheken über Anaconda verwaltbar (installieren mit *conda install*). Sie können den Code vermutlich auch mit anderen, aktuelleren Versionen von Python und den genannten Bibliotheken ausführen. Allerdings ist es möglich, dass Sie dann an

der einen oder anderen Stelle auf Warnungen, Fehlermeldungen oder auf andere Probleme stoßen, die wir nicht vorhersehen können.

Die Codebeispiele im Buch sind über *GitHub* als *Jupyter Notebooks* verfügbar. Sie können den Code entweder im Internet unter *https://github.com/tplusone/hanser_deep_nlp* abrufen und ansehen oder, wenn Sie die Software *Git* installiert haben, das gesamte Repository inklusive Codebeispielen und Beispieldaten über das Terminal mit dem folgenden Befehl auf Ihren Rechner ziehen:

```
git clone https://github.com/tplusone/hanser_deep_nlp.git
```

---

[1] Vgl. *https://www.wired.com/story/self-driving-cars-challenges/* [Abgerufen am 11.09.2021]

2 Textdaten verarbeiten und vorverarbeiten

Wenn es um die Verarbeitung numerischer und textbasierter Informationen geht, hat Python gegenüber anderen Programmiersprachen ein paar entscheidende Vorteile. Schon die Standardbibliothek bietet eine Vielzahl einfacher Funktionen und Klassen, um Textdaten in Form zu bringen, zu transformieren und zu strukturieren. So ist es ein Leichtes, einmal tokenisierte Texte in Arrays zu verwalten, Auszüge zu extrahieren oder mithilfe von Schleifen oder List Comprehensions Transformationen durchzuführen. Auch Casting-Operationen, die Arrays in Sets, Tuples oder Dictionarys verwandeln, funktionieren in den meisten Fällen vorhersehbar und problemlos. So lassen sich Texte für das Anlernen in Form bringen.

Um einen maschinellen Lernalgorithmus zu trainieren, brauchen wir aber andere Datenklassen. Statistische Verfahren operieren mit mathematischen Funktionen und mögen daher keine Überraschungen, wenn es um Datentypen und die Struktur der Datenbehälter geht. Da eines der Grundprinzipien in Python aber die Abkehr von Typsicherheit zugunsten von Duck-Typing ist, benötigen wir eine Alternative, die dieses Anforderungsprofil erfüllt. Die Bibliothek *NumPy* (Numerical Python) ist dabei die erste Wahl. NumPy ist zum Glück auf die Datenklassen der Standardbibliothek abgestimmt, sodass Castings in beide Richtungen reibungslos funktionieren.

Im Folgenden sehen wir uns einige ausgewählte Techniken, Klassen und Bibliotheken an, die für die Vorverarbeitung von Texten zur Analyse mit Lernalgorithmen von Bedeutung sind. Dabei handelt es sich um keine auch nur annähernd vollständige Abhandlung der verschiedenen Möglichkeiten. Wir werfen aber einen Blick auf die Verfahren, die in den nachfolgenden Kapiteln immer wieder verwendet werden und die wir dort nicht noch einmal im Detail vorstellen. Neben der Tokenisierung und numerischen Encodierung von Wörtern geht es um die Repräsentation von Wörtern als One-Hot-Sets und natürlich um die Arbeit mit der Bibliothek NumPy.