O'REILLY®

Deutsche Ausgabe

PyTorch für Deep Learning

Anwendungen für Bild-, Ton- und Textdaten entwickeln und deployen

Ian Pointer

Übersetzung von Marcus Fraaß

PyTorch für Deep Learning

Anwendungen für Bild-, Ton- und Textdaten entwickeln und deployen

Ian Pointer

Deutsche Übersetzung von Marcus Fraaß

O'REILLY®

Ian Pointer

Lektorat: Alexandra Follenius
Übersetzung: Marcus Fraaß
Korrektorat: Sibylle Feldmann, *www.richtiger-text.de*
Satz: III-satz, *www.drei-satz.de*
Herstellung: Stefanie Weidner und Frank Heidt
Umschlaggestaltung: Karen Montgomery, Michael Oréal, *www.oreal.de*

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über *http://dnb.d-nb.de* abrufbar.

ISBN:
Print 978-3-96009-134-9
PDF 978-3-96010-399-8
ePub 978-3-96010-400-1
mobi 978-3-96010-401-8

1. Auflage 2021

Wieblinger Weg 17
69123 Heidelberg

Dieses Buch erscheint in Kooperation mit O'Reilly Media, Inc. unter dem Imprint »O'REILLY«. O'REILLY ist ein Markenzeichen und eine eingetragene Marke von O'Reilly Media, Inc. und wird mit Einwilligung des Eigentümers verwendet.

Hinweis:
Dieses Buch wurde auf PEFC-zertifiziertem Papier aus nachhaltiger Waldwirtschaft gedruckt. Der Umwelt zuliebe verzichten wir zusätzlich auf die Einschweißfolie.

Schreiben Sie uns:
Falls Sie Anregungen, Wünsche und Kommentare haben, lassen Sie es uns wissen: kommentar@oreilly.de.

5 4 3 2 1 0

Inhalt

Index

Vorwort

Deep Learning in der heutigen Zeit

Hallo und herzlich willkommen! Dieses Buch wird Sie in das Thema Deep Learning mit PyTorch, einer Open-Source-Bibliothek, die im Jahr 2017 von Facebook öffentlich bereitgestellt wurde, einführen. Wenn Sie in den letzten Jahren nicht wie ein Vogel Strauß mit dem Kopf im Sand gesteckt haben, werden Sie nicht umhingekommen sein, zu bemerken, dass neuronale Netzwerke heutzutage überall zugegen sind. Sie sind von dem »wirklich coolen Teil der Informatik, über den die Menschen lernen und danach nichts damit anfangen können« dazu gereift, jeden Tag in unseren Smartphones mit uns herumgetragen zu werden, um unsere Bilder zu verbessern oder unseren Sprachbefehlen zuzuhören. Unsere E-Mail-Programme lesen unsere E-Mails und produzieren kontextbezogene Antworten, unsere Lautsprecher hören uns zu, Autos fahren von selbst, und der Computer hat die besten Spieler bei Go inzwischen besiegt. Wir sehen auch, dass die Technologie in autoritären Ländern für ruchlosere Zwecke eingesetzt wird, etwa wenn Wächter mithilfe von neuronalen Netzwerken Gesichter aus der Menge herauspicken und darüber entscheiden, ob die Menschen verhaftet werden sollen.

Auch wenn es einem so vorkommen mag, dass dies alles sehr rasch voranging, so reichen die zugrunde liegenden

Konzepte der neuronalen Netze und des Deep Learning dennoch weit zurück. Der Beweis, dass ein solches Netz in der Lage ist, *jede* mathematische Funktion in einer approximativen Weise zu ersetzen – was die Idee untermauert, dass neuronale Netze für viele verschiedene Aufgaben trainiert werden können –, geht zurück ins Jahr 1989.[1] Neuronale Konvolutionsnetze wurden bereits in den späten 1990er-Jahren zur Erkennung von Ziffern auf Schecks verwendet. In dieser Zeit hat sich ein solides Fundament aufgebaut. Warum also fühlt es sich so an, als hätte es in den letzten zehn Jahren eine explosionsartige Entwicklung in diesem Bereich gegeben?

Hierfür gibt es zahlreiche Gründe, wobei der wichtigste wohl der Anstieg der Leistungsfähigkeit von *Grafikprozessoren* (GPUs) und deren zunehmende Erschwinglichkeit ist. Ursprünglich für Spiele entwickelt, sind GPUs darauf ausgelegt, unzählige Millionen von Matrixoperationen pro Sekunde auszuführen, um alle Polygone für das Fahr- oder Ballerspiel, das Sie auf Ihrer Konsole oder Ihrem PC spielen, zu rendern – Operationen, für die ein gewöhnlicher Hauptprozessor (CPU) einfach nicht optimiert ist. Der im Jahr 2009 erschienene Forschungsbeitrag »Large-Scale Deep Unsupervised Learning Using Graphics Processors« von Rajat Raina et al.[2] wies darauf hin, dass das Training neuronaler Netze ebenfalls auf der Ausführung zahlreicher Matrixoperationen basiert, sodass zusätzliche Grafikkarten das Training beschleunigen und dadurch auch erstmals noch größere, *tiefere* neuronale Netzwerkarchitekturen eingesetzt werden können.

Andere wichtige Methoden wie *Dropout* (die wir in Kapitel 3 betrachten werden) wurden ebenfalls in den letzten zehn Jahren eingeführt, um das Training nicht nur zu

beschleunigen, sondern auch besser zu *verallgemeinern* (damit das Netzwerk nicht einfach nur die Trainingsdaten auswendig lernt – ein Problem namens *Überanpassung*, auf das wir im nächsten Kapitel stoßen werden). In den letzten Jahren haben Unternehmen diesen GPU-basierten Ansatz auf die nächste Ebene gehoben. Dabei hat Google etwas entwickelt, das es als *Tensorprozessoren* (TPUs) bezeichnet. Es handelt sich dabei um anwendungsspezifische Chips, die speziell dafür entwickelt wurden, so schnell wie möglich Deep-Learning-Anwendungen durchzuführen. Darüber hinaus stehen sie sogar der allgemeinen Öffentlichkeit als Teil des Google-Cloud-Ökosystems zur Verfügung.

Eine weitere Möglichkeit, den Fortschritt des Deep Learning in den letzten zehn Jahren zu skizzieren, bietet der ImageNet-Wettbewerb. ImageNet ist eine riesige Datenbank mit über 14 Millionen Bildern, die zu Zwecken des maschinellen Lernens für 20.000 Kategorien manuell gelabelt sind. Seit dem Jahr 2010 wird jährlich im Rahmen des Wettbewerbs *ImageNet Large Scale Visual Recognition Challenge* versucht, die Anwendungen aller Teilnehmer anhand eines Auszugs aus der Datenbank mit 1.000 Kategorien zu testen. Bis 2012 lagen die Fehlerquoten für die Bewältigung der Aufgaben bei etwa 25%. In jenem Jahr gewann jedoch ein tiefes neuronales Konvolutionsnetz den Wettbewerb mit einer Fehlerquote von 16%, womit es alle anderen Teilnehmer deutlich übertraf. In den folgenden Jahren wurde diese Fehlerquote immer weiter nach unten gedrückt, bis die ResNet-Architektur im Jahr 2015 ein Ergebnis von 3,6% erreichte, das unterhalb der durchschnittlichen menschlichen Leistung (5%) lag. Wir wurden also überholt.

Aber was genau ist Deep Learning, und brauche ich einen Doktortitel, um es verstehen zu können?

Die Definitionen von Deep Learning sind oft eher verwirrend als erhellend. Eine Definitionsmöglichkeit besteht darin, Deep Learning als eine Methode des maschinellen Lernens zu bezeichnen, die mehrere und zahlreiche Schichten nicht linearer Transformationen verwendet, um nach und nach Merkmale aus der ursprünglichen Eingabe zu extrahieren. Das ist zwar richtig, aber es hilft nicht wirklich weiter, oder? Ich ziehe es vor, es als eine Methode zu beschreiben, die Aufgaben löst, bei denen Sie dem Computer die Eingaben und die gewünschten Ausgaben liefern und er die Lösung findet, im Regelfall unter Verwendung eines neuronalen Netzes.

Auf viele Menschen abschreckend wirkt im Zusammenhang mit Deep Learning die dahinterstehende Mathematik. Wenn Sie sich irgendeinem Forschungspapier in diesem Bereich widmen, werden Sie fast überall auf undurchdringliche Mengen an Notationen in Form von griechischen Buchstaben stoßen und wahrscheinlich schreiend im Dreieck springen. Die Sache ist die: Man muss in den meisten Fällen kein Mathegenie sein, um die Methoden des Deep Learning anwenden zu können. Tatsächlich muss man für die meisten alltäglichen, grundlegenden Anwendungen der Technologie überhaupt nicht viel wissen. Um wirklich zu verstehen, was vor sich geht (wie Sie in Kapitel 2 sehen werden), muss man sich nur ein wenig anstrengen, um die Konzepte nachvollziehen zu können, die man wahrscheinlich bereits in der Oberstufe gelernt hat. Lassen Sie sich also nicht von der Mathematik verunsichern. Am Ende von Kapitel 3 werden Sie in der

Lage sein, einen Bildklassifikator zusammenzustellen, der mit nur wenigen Zeilen Code mit dem konkurriert, was die besten Köpfe im Jahr 2015 anbieten konnten.

PyTorch

Wie bereits zu Beginn erwähnt, ist PyTorch eine Open-Source-Bibliothek von Facebook, die das Programmieren von Deep-Learning-Code in Python erleichtert. Der Ursprung der Bibliothek fußt wiederum auf zwei weiteren Bibliotheken. Sie leitet, und angesichts ihres Namens vielleicht nicht ganz überraschend, viele Funktionen und Konzepte von *Torch* ab, einer Lua-basierten Bibliothek für neuronale Netzwerke, die auf das Jahr 2002 zurückgeht. Der zweite wichtige Vorläufer ist die Bibliothek *Chainer*, die im Jahr 2015 in Japan entwickelt wurde. Chainer war eine der ersten Bibliotheken für neuronale Netzwerke, die einen eifrigen bzw. interaktiven (engl. Eager Execution) Ansatz zur Differenzierung anstelle der Definition statischer Graphen bot. Das ermöglichte eine größere Flexibilität bei der Erstellung, dem Training und der Handhabung von Netzwerken. Die Kombination aus Elementen von Torch und Ideen von Chainer hat PyTorch in den letzten Jahren populär gemacht.[3]

Die Bibliothek umfasst auch hilfreiche Module zur Text-, Bild- und Tondatenmanipulation (`torchtext`, `torchvision` und `torchaudio`) sowie integrierte Varianten populärer Architekturen wie ResNet (mit Gewichten, die heruntergeladen werden können, um bei Methoden wie *Transfer Learning*, das Sie in Kapitel 4 kennenlernen werden, Unterstützung zu bieten).

Auch über Facebook hinaus gewann PyTorch in der Industrie schnell an Akzeptanz. Unternehmen wie Twitter,

Salesforce, Uber und NVIDIA nutzen es auf verschiedene Weise für ihre Deep-Learning-Anwendungen. Ich ahne aber bereits die nächste Frage …

Warum nicht TensorFlow?

Genau, richten wir unseren Blick auf das ziemlich große von Google konzipierte Schwergewicht in der gegenüberliegenden Ecke. Was bietet PyTorch, was TensorFlow nicht bieten kann? Warum sollten Sie stattdessen PyTorch lernen?

Die Antwort ist, dass das traditionelle TensorFlow anders funktioniert als PyTorch, was erhebliche Auswirkungen auf den Code und die Fehlersuche mit sich bringt. In TensorFlow verwenden Sie die Bibliothek, um eine Graphendarstellung der Architektur des neuronalen Netzwerks aufzubauen, und führen dann Operationen auf diesem Graphen aus, was innerhalb der TensorFlow-Bibliothek geschieht. Diese Methode der deklarativen Programmierung steht im Widerspruch zum imperativen Paradigma von Python, was bedeutet, dass TensorFlow-Programme in Python etwas seltsam und schwer verständlich aussehen und sich auch so anfühlen können. Das andere Problem ist, dass im Vergleich zum Ansatz mit PyTorch die statische Graphendeklaration die dynamische Änderung der Architektur während der Trainings- und Inferenzphase sehr viel komplizierter macht und mit Boilerplate-Code vollstopfen kann.

Aus diesen Gründen ist PyTorch in den forschungsorientierten Fachkreisen populär geworden. Die Anzahl der bei der »International Conference on Learning Representations« eingereichten Beiträge, in denen *PyTorch* erwähnt wird, ist im letzten Jahr um 200% gestiegen. Die

Anzahl der Beiträge, in denen *TensorFlow* erwähnt wird, hat fast genauso stark zugenommen. PyTorch wird mit Sicherheit auch in Zukunft eine Rolle spielen.

Mit dem Erscheinen neuerer Versionen von TensorFlow änderten sich die Dinge jedoch ein wenig. Eine neue Funktion namens *Eager Execution*, die es ermöglicht, ähnlich wie mit PyTorch zu arbeiten, wurde unlängst der Bibliothek hinzugefügt. Das wird auch das Paradigma sein, das in TensorFlow 2.0 vorangetrieben wird. Da jedoch nur wenige neue Informationsmaterialien außerhalb von Google zur Verfügung stehen, die Ihnen helfen, diese neue Arbeitsweise mit TensorFlow zu erlernen, sind die Möglichkeiten sehr begrenzt. Sie müssten jahrelang daran arbeiten, um das andere Paradigma zu verstehen, damit Sie das Beste aus der Bibliothek herausholen können.

Aber nichts davon sollte Ihnen ein schlechtes Bild von TensorFlow vermitteln; es bleibt eine industrieerprobte Bibliothek, die durch eines der größten Unternehmen der Welt gefördert wird. PyTorch (natürlich ebenfalls gestützt durch eine andere »größte Firma der Welt«) ist, würde ich sagen, ein rationalisierter und fokussierterer Ansatz für Deep Learning und differenzielle Programmierung. Da keine älteren, eingestaubten APIs mehr unterstützt werden müssen, ist es leichter, in PyTorch zu lehren und produktiv zu werden als in TensorFlow.

Wie hängt Keras damit zusammen? So viele gute Fragen! Keras ist eine übergeordnete Deep-Learning-Bibliothek, die ursprünglich Theano und TensorFlow und nun auch andere Frameworks wie Apache MXNet unterstützt. Sie bietet bestimmte Funktionen wie Trainings-, Validierungs- und Testroutinen, deren Erstellung die untergeordneten Frameworks sonst dem Entwickler überlassen, sowie einfache Methoden zum Aufbau von neuronalen

Netzwerkarchitekturen. Sie hat enorm zur Verbreitung von TensorFlow beigetragen, ist jetzt Teil von TensorFlow selbst (als `tf.keras`) und wird auch weiterhin ein eigenständiges Projekt bleiben. Im Vergleich dazu ist PyTorch so etwas wie der Mittelweg zwischen reinem TensorFlow und Keras; wir werden unsere eigenen Trainings- und Vorhersageroutinen schreiben müssen, aber das Erstellen von neuronalen Netzwerken ist fast genauso einfach (und meiner Meinung nach ist der Ansatz von PyTorch zur Erstellung und Wiederverwendung von Architekturen für einen Python-Entwickler bedeutend logischer als der bei Keras).

Wie Sie in diesem Buch noch sehen werden, ist PyTorch seit der Einführung von Version 1.0, obwohl es in eher forschungsorientierten Kreisen verbreitet ist, perfekt für Anwendungsfälle im Produktiveinsatz geeignet.

Ziel und Ansatz

Das vorliegende Buch führt Sie in die Grundlagen von PyTorch ein und zeigt Ihnen, wie Sie selbst Ihre eigenen Deep-Learning-Anwendungen Schritt für Schritt entwickeln, debuggen und in den Produktivbetrieb überführen können. Sie lernen, wie man neuronale Netzwerke in PyTorch erstellt, schrittweise komplexere Modellelemente integriert und schließlich auch Modelle nutzt, die den neuesten Stand der Technik abbilden. Sie erfahren, wie Sie Ihren eigenen Datensatz aufbauen und erweitern können, und auch, wie Sie sich Transfer Learning zunutze machen, wenn Sie nur auf wenige Daten zurückgreifen können oder die Leistungsfähigkeit Ihres Modells weiter verbessern möchten.

Darüber hinaus lernen Sie, wie Sie Deep-Learning-Modelle erfolgreich überwachen und debuggen. Sie werden auch sehen, wie Sie die Entscheidungsfindung eines Modells mithilfe von Visualisierungen offenlegen können. Sie erfahren, wie sich Ihre Anwendung als skalierbarer Webdienst unter Nutzung von Docker, Kubernetes und Google Cloud in den Produktivbetrieb überführen lässt. Zudem sehen wir uns verschiedene Ansätze dazu an, wie wir bereits trainierte Modelle einerseits komprimieren und andererseits auch das Modelltraining, beispielsweise durch die direkte oder indirekte Nutzung der Programmiersprache C++ oder einer GPU, beschleunigen können.

Wir trainieren unsere Modelle mit Daten aus drei verschiedenen Domänen: Bilder, Texte und Töne. Dabei lernen Sie, in welche Form Sie diese Datentypen zur weiteren Verarbeitung bringen müssen, welche Unterschiede Sie hinsichtlich der Modellwahl beachten sollten, welche Modellarchitekturen sich für welche Domäne besonders anbieten, und sogar, wann es sich lohnt, in eine andere Domäne mithilfe einer Transformation der Daten zu wechseln. Dabei erfahren Sie, wie Sie in PyTorch sowohl integrierte Modelle und Klassen nutzen als auch eigenständig Erweiterungen oder Modifizierungen umsetzen können.

Im Verlauf des Buchs begegnen Ihnen zahlreiche hilfreiche Tipps und Tricks, die Ihnen das Leben bei der täglichen Anwendung von PyTorch erleichtern sollen. Das Buch soll Ihnen nicht nur einen Leitfaden an die Hand geben, sondern Sie auch zu selbstständigem Lernen anregen und Ihnen aufzeigen, was PyTorchs Ökosystem sonst noch für Sie bereithält. Scheuen Sie sich also nicht, zu gegebener Zeit einen Blick in die Dokumentationen oder in den

Quellcode der genutzten Bibliotheken und Pakete zu werfen – es lohnt sich ungemein!

Voraussetzungen

Die Hauptzielgruppe dieses Buchs sind Entwickler und Data Scientists, die gegebenenfalls noch wenige Kenntnisse im Bereich Deep Learning haben, aber mehr darüber lernen möchten. Auch wenn Vorkenntnisse wie z.B. ein grundlegendes Verständnis für die Funktionsweise von neuronalen Netzwerken und deren Optimierungsroutinen hilfreich sind, so sind sie nicht unbedingt erforderlich.

Darüber hinaus richtet sich das Buch an diejenigen, die bereits Erfahrungen in der Programmierung im Bereich Deep Learning haben und auf PyTorch umsteigen oder es noch eingehender lernen möchten. Sie sollten über eine gewisse Erfahrung in der Programmierung mit Python oder anderen Programmiersprachen (z.B. Java, Scala, Ruby, R usw.) verfügen und sich bereits ein wenig mit der Handhabung der Kommandozeile auskennen.

Wenn Sie mit Python noch nicht vertraut sind, finden Sie einen guten Einstieg auf *https://www.learnpython.org*. Darüber hinaus gibt es unzählige kostenlose Onlineressourcen, die es Ihnen ermöglichen, sich genügend Python-Wissen anzueignen, um mit den Beispielen in diesem Buch arbeiten zu können.

Der Fokus des Buchs liegt auf der eigenständigen Entwicklung und Umsetzung von Deep-Learning-Modellen in PyTorch. Dabei wird weitestgehend auf mathematische Notationen und komplexe Formeln verzichtet. Sie müssen dementsprechend auch nicht über einen nennenswerten mathematischen Hintergrund verfügen, um dem Buch folgen zu können.

Sie benötigen eine Rechnerumgebung, in der Sie die verwendeten Codebeispiele, die Sie auch in dem GitHub-Repository (*https://oreil.ly/pytorch-github*) des Buchs finden, ausführen können. Das Repository können Sie mit dem Befehl `git clone https://github.com/falloutdurham/beginners-pytorch-deep-learning.git` herunterladen (oder besser, Sie forken zunächst das Repository und laden es dann von Ihrem eigenen Account herunter) und die Beispiele in Ihrer gewünschten Entwicklungsumgebung ausführen (ggf. müssen Sie noch die Pfadangaben individuell an Ihre eigene Verzeichnisstruktur anpassen). In Kapitel 1 werden Sie noch ausführlich erfahren, wie Sie Ihre Entwicklungsumgebung einrichten können.

Wegweiser durch dieses Buch

In Kapitel 1, *Einstieg in PyTorch*, richten wir alles ein, um mit der Programmierung in PyTorch loslegen zu können. Hierbei zeige ich Ihnen, wie Sie PyTorch sowohl lokal als auch in der Cloud nutzen können, und ich gehe auch kurz auf die grundlegendsten Elemente und Befehle der Bibliothek ein.

In Kapitel 2, *Bildklassifizierung mit PyTorch*, lernen Sie die Grundlagen neuronaler Netze kennen und erfahren, wie Sie ein neuronales Netzwerk in PyTorch selbst programmieren, das zur Bildverarbeitung bereitstehende Paket `torchvision` verwenden und Ihr Modell zur Bildklassifizierung nutzen können. Hierfür entwickeln wir zunächst ein relativ einfaches, vollständig verbundenes neuronales Netz, einen eigenen Datensatz mit Bildern aus dem Internet und eine Trainingsroutine, mit der wir das neuronale Netz auf der GPU trainieren. Zudem sehen wir uns an, wie Sie Vorhersagen für Bilder vornehmen und wie Sie Modelle auf

und von der Festplatte speichern bzw. wiederherstellen können. Bis einschließlich Kapitel 4 entwickeln wir davon ausgehend unterschiedliche Netzwerke, mit denen wir Bilder klassifizieren und feststellen möchten, ob sich auf einem Bild eine Katze oder ein Fisch befindet. Dabei lernen Sie die wichtigsten Bestandteile und Parameter neuronaler Netzwerke kennen, wie Aktivierungs- und Verlustfunktionen, Lernrate sowie Optimierungsalgorithmen.

Im weiteren Verlauf erhöhen wir kontinuierlich die Leistungsfähigkeit unserer Netzwerke. In Kapitel 3 lernen Sie mit den neuronalen Konvolutionsnetzen (engl. Convolutional Neural Networks, CNNs) eine weitere Architektur kennen, und Sie werden beobachten, dass sich CNNs schneller trainieren lassen und auch genauer sind. In diesem Zusammenhang zeige ich Ihnen verschiedene neuronale Schichttypen und deren Funktionsweise. Zudem gebe ich Ihnen einen kurzen geschichtlichen Einblick in die Entwicklung der bemerkenswertesten Netzwerkarchitekturen im Bereich des bildbasierten Deep Learning. Anschließend werfen wir einen Blick darauf, wie man bereits zuvor trainierte Modelle in PyTorch nutzen kann.

In Kapitel 4, *Transfer Learning und andere Kniffe*, vertiefen wir das Thema Bildverarbeitung noch weiter und greifen dabei auf eine unglaublich effektive Technik im Deep Learning namens Transfer Learning zurück. Hierbei wird ein Modell auf eine Aufgabe (vor-)trainiert und im Anschluss daran auf eine andere Aufgabe übertragen, weshalb es sich sehr schnell und äußerst effektiv in Anwendung bringen lässt. In diesem Zusammenhang greifen wir auf ein großes, auf einen anderen Datensatz vortrainiertes Netzwerk namens ResNet zurück und

nehmen darauf basierend eine Feinabstimmung auf unseren Datensatz vor, um unsere Katze-oder-Fisch-Klassifizierung noch akkurater zu gestalten. Hierbei zeige ich Ihnen auch, wie Sie die optimale Lernrate finden, differenzielle Lernraten im Rahmen der Feinabstimmung heranziehen und auch Datenaugmentationstechniken sowie Ensemble-Modelle zur weiteren Verbesserung Ihres Modells nutzen können.

In Kapitel 5, *Textklassifizierung*, lassen wir die Bilddomäne vorerst hinter uns und widmen uns textbasierten Ansätzen des Deep Learning zur Klassifizierung. Das Kapitel bietet einen Schnelleinstieg in die Thematik rund um rekurrente neuronale Netzwerke und deren Erweiterungen. Wir sehen uns an, wie Textdaten in Form von Einbettungen (engl. Embeddings) codiert und für das Deep Learning nutzbar gemacht werden. Außerdem erkunden wir das `torchtext`-Paket sowie die Erstellung eines Twitter-Datensatzes und wie man das Paket im Rahmen der Textverarbeitung für die Entwicklung eines LSTM-basierten Modells zur Klassifizierung von Tweets nutzt. Darüber hinaus sehen wir uns ebenso für Textdaten Strategien an, die Ihnen helfen, Ihren Datensatz zu erweitern.

Kapitel 6, *Eine Reise in die Welt der Klänge*, bietet Ihnen einen Einstieg in die Verarbeitung von Ton- bzw. Audiodaten. Wir sehen uns zwei sehr unterschiedliche Ansätze zur Klassifizierung an: einen, der auf der reinen Wellenform der Audiodaten und einem neuronalen Konvolutionsnetz beruht, und einen anderen, bei dem wir die Audiodaten zuvor in den Frequenzbereich überführen und uns erneut Transfer Learning zunutze machen. Dabei unternehmen wir einen kurzen Einstieg in das `torchaudio`-Paket und sehen uns an, wie sich Transformationen von Datensätzen effektiv berechnen lassen. Außerdem widmen

wir uns abschließend mehreren Ansätzen zur Datenaugmentation von Audiodaten, die die Leistungsfähigkeit der Modelle und ihre Fähigkeit, zu verallgemeinern, verbessern sollen.

In Kapitel 7, *PyTorch-Modelle debuggen*, kommen wir darauf zu sprechen, wie man Modelle debuggen kann, wenn das Training nicht korrekt oder nicht schnell genug vonstattengeht. Wir werfen einen Blick darauf, wie wir mit sogenanntem Class Activation Mapping unter Nutzung von PyTorch-Hooks Deep-Learning-Modelle nachvollziehbarer gestalten können und wie man PyTorch zu Debugging-Zwecken mit Googles TensorBoard verbindet. Zudem erfahren Sie, wie Flammendiagramme eingesetzt werden können, um rechenintensive Funktionsaufrufe zu identifizieren und zu beseitigen, speziell in Bezug auf mögliche Verzögerungen bei Transformationen. Schließlich sehen wir uns an, wie Sie die Auslastung Ihrer GPU im Blick behalten, den Speicherbedarf reduzieren und wie Sie insbesondere bei der Verwendung größerer Modelle mithilfe von Checkpoints mehr Speicherkapazität auf Ihrer GPU erlangen können.

In Kapitel 8, *PyTorch im Produktiveinsatz*, kommen wir darauf zu sprechen, wie Sie PyTorch-Anwendungen in eine Produktionsumgebung überführen, das Deployment des Modells gestalten und Ihr Modell weiter optimieren können. Wir entwickeln einen skalierbaren Webdienst, mit dem wir in PyTorch erstellte Vorhersagemodelle über HTTP und gRPC bereitstellen. Dabei packen wir unsere Anwendung in einen Docker-Container, stellen sie in einem Kubernetes-Cluster über Google Cloud bereit und führen uns vor Augen, wie wir die Anwendung mittels Telemetrie und Logging überwachen können. Im zweiten Teil beschäftigen wir uns mit Optimierungsmöglichkeiten

unseres Codes. Wir erkunden in diesem Zusammenhang TorchScript, das uns aufgrund seiner statischen Typisierung und graphenbasierten Darstellung ermöglicht, Python-basierte Modelle weiter zu optimieren. Ich zeige Ihnen, wie Sie mithilfe von Just-in-Time-(JIT-)Tracing und Scripting optimierte und Python-kompatible TorchScript-Modelle erstellen können, die überdies mithilfe von libTorch in C++ ausgeführt und dadurch noch weiter optimiert werden können. Abschließend werfen wir auch einen kurzen Blick darauf, wie große Modelle mittels Quantisierung komprimiert werden können.

In Kapitel 9, *Praxiserprobte PyTorch-Modelle in Aktion*, sehen wir uns an, wie PyTorch von Personen und Unternehmen rund um den Globus genutzt wird und wie Sie selber diese hochaktuellen Modelle mit bereits zur Verfügung stehenden Bibliotheken für Ihre Anwendung nutzen können. Dabei lernen Sie einige neue Ansätze und Modelle kennen. Zuerst beschäftigen wir uns damit, wie Sie Ihr Modell mithilfe weiterer Datenaugmentationstechniken noch verbessern und einer Überanpassung vorbeugen können. Anschließend skizziere ich Ihnen den Aufbau von Super-Resolution-Modellen, die auf Generative Adversarial Networks (GANs) basieren und mit denen wir Bilder hochauflösend hochskalieren können. Im Anschluss daran zeige ich Ihnen, wie Sie relativ schnell mit dem Konvolutionsnetzwerk Faster R-CNN Objekte auf Bildern erkennen und wie Sie Bilder erstellen können, die sogar dazu in der Lage sind, neuronale Netzwerke zu täuschen – sogenannte Adversarial Samples. Wir schauen uns die Grundstruktur von Transformer-basierten Architekturen an und wie diese insbesondere im Bereich des Natural Language Processing als Sprachmodelle zum Tragen kommen. Hierbei zeige ich Ihnen, wie Sie Ihre Feinabstimmung mit einem Transformer-basierten Modell

vornehmen (FastBERT), synthetische Tweets mit GPT-2 erzeugen und ein vortrainiertes ULMFiT-Modell sehr schnell mithilfe der fast.ai-Bibliothek zur Klassifizierung von Tweets heranziehen können, das zwar nicht auf dieser Architektur basiert, aber dennoch zufriedenstellende Ergebnisse liefert. Zu guter Letzt greifen wir noch auf PyTorch Lightning zurück, und ich zeige Ihnen verschiedene Anwendungsfälle des selbstüberwachten Lernens im Rahmen des Vorabtrainings, mit dem sich die Modellleistung für verschiedene Aufgabenstellungen weiter verbessern lässt. Hier lernen Sie einerseits den Ansatz kennen, Eingaben zu verändern und diese mithilfe von Selbstüberwachung wiederherzustellen, und andererseits den vielversprechenden Ansatz, Labels für Ihre Daten ohne jeglichen menschlichen Annotationsaufwand zu erstellen.

Veränderungen gegenüber der englischsprachigen Ausgabe

Die vorliegende deutsche Übersetzung unterscheidet sich an wenigen Stellen gegenüber dem englischsprachigen Original. Zum einen wurde der abgedruckte Code aktualisiert, um die Kompatibilität zu neueren Versionen der genutzten Bibliotheken zu gewährleisten, und Errata wurden ausgebessert. Zum anderen wurden der deutschsprachigen Ausgabe in enger Zusammenarbeit mit dem Autor Ian Pointer neue Abschnitte hinzugefügt oder überarbeitet. Dies betrifft den Abschnitt zur Quantisierung von Modellen zum Ende des Kapitels 8 und den Abschnitt zum selbstüberwachten Lernen zum Abschluss des Kapitels 9. Der Abschnitt zu den Transformer-basierten Modellen (ULMFiT, GPT-2 und FastBERT) wurde überarbeitet, und die Codebeispiele wurden komplett in PyTorch überführt.

Darüber hinaus wurde der Abschnitt »Alles in einem« in Kapitel 2 inhaltlich ergänzt.

Softwareversionen

Der Code in diesem Buch wurde mit den folgenden Versionen auf Funktionsfähigkeit getestet: Python 3.6.9, PyTorch (torch) 1.6.0 (bis auf den Code im Abschnitt »Faster R-CNN und Mask R-CNN« auf Seite 203, der PyTorch-Version 1.0 nutzt), torchvision 0.7.0, torchtext 0.7.0 und torchaudio 0.6.0. Auf der GitHub-Seite des Buchs finden Sie Dateien mit einer vollständigen Auflistung aller genutzten Versionen. Diese können Sie zur Versionskontrolle nutzen, insbesondere dann, wenn Sie die Installationen lokal vornehmen möchten und in einer eigens dafür vorgesehenen virtuellen Umgebung arbeiten.

Sollten Sie mit dem Paketverwaltungsprogramm `pip` arbeiten, empfehle ich Ihnen, mit dem Befehl `python3 -m venv .venv` von Ihrer Kommandozeile aus (wenn Sie sich im Zielordner befinden) eine virtuelle Umgebung einzurichten (*https://docs.python.org/3/library/venv.html#module-venv*). Nach Aktivierung der virtuellen Umgebung können Sie auf Basis der Datei *requirements.txt* alle notwendigen Bibliotheken und Pakete mit dem Befehl `pip install -r requirement.txt` installieren. Wenn Sie sich für `conda` entscheiden, steht Ihnen eine Datei namens *environment.yml* zur Verfügung, mit der Sie die Installationen mit dem Befehl `conda env create` durchführen können. Auf gleiche Weise können Sie auch die Version bei der Installation einzelner Pakete kontrollieren. Beispielsweise könnten Sie die Befehle `pip install torch==1.6.0` oder `conda install torch==1.6.0` ausführen, um die Installation von PyTorch-Version 1.6.0 zu gewährleisten.

Es ist nicht auszuschließen, dass Veränderungen an den zugrunde liegenden Bibliotheken oder Paketen vorgenommen werden oder dass sich Schnittstellendefinitionen verändern. Sollten Sie einmal feststellen, dass der Code nicht wie gewünscht läuft, zögern Sie nicht, im Repository auf GitHub (*https://oreil.ly/pytorch-github*) nachzusehen, ob neuere Codeversionen verfügbar sind (oder gesonderte Branches) oder ob es offene oder geschlossene Issues gibt, die Ihnen bei Ihrem Problem weiterhelfen könnten.

Weitere Ressourcen

- *Deep Learning – Grundlagen und Implementierung: Neuronale Netze mit Python und PyTorch programmieren* von Seth Weidman, O'Reilly 2020
- *Natural Language Processing mit PyTorch: Intelligente Sprachanwendungen mit Deep Learning erstellen* von Delip Rao und Brian McMahan, O'Reilly 2019
- *GANs mit PyTorch selbst programmieren: Ein verständlicher Einstieg in Generative Adversarial Networks* von Tariq Rashid, O'Reilly 2020

In diesem Buch verwendete Konventionen

Die folgenden typografischen Konventionen werden in diesem Buch eingesetzt:

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, Umgebungsvariablen, 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.

Dieses Symbol steht für einen Tipp oder eine Empfehlung.

Dieses Symbol steht für einen allgemeinen Hinweis.

Dieses Symbol warnt oder mahnt zur Vorsicht.

Verwenden von Codebeispielen

Zusätzliche Materialien (Codebeispiele, Übungen und so weiter) können Sie von der Adresse *https://oreil.ly/pytorch-github* herunterladen.