Natural Language Processing mit Transformern

Sprachanwendungen mit Hugging Face erstellen

KAPITEL 1
Hallo Transformer

Im Jahr 2017 veröffentlichten Forscher von Google einen Artikel, der eine neuartige neuronale Netzwerkarchitektur für die Modellierung von Sequenzen vorschlug.[1] Diese als *Transformer* bezeichnete Architektur übertraf rekurrente neuronale Netze (engl. Recurrent Neural Networks, RNNs) bei maschinellen Übersetzungsaufgaben sowohl in Bezug auf die Übersetzungsqualität als auch auf die Trainingskosten.

Gleichzeitig wurde mit einem effektiven Transfer-Learning-Verfahren namens ULMFiT gezeigt, dass durch das Training von LSTM-Netzwerken (Long Short-Term Memory) auf einem sehr großen und vielfältigen Korpus hochmoderne Textklassifikatoren erstellt werden können, die nur wenige gelabelte Daten erfordern.[2]

Diese Fortschritte bildeten die Grundlage für zwei der heute bekanntesten Transformer-Modelle: zum einen dem Generative Pretrained Transformer (GPT)[3] und zum anderen Bidirectional Encoder Representations from Transformers (BERT)[4]. Durch die Kombination der Transformer-Architektur mit unüberwachtem Lernen (engl. Unsupervised Learning) machten diese Modelle das Training aufgabenspezifischer Architekturen, die von Grund auf trainiert werden müssen, überflüssig und brachen fast jede Benchmark im NLP-Bereich mit deutlichem Abstand. Seit der Veröffentlichung von GPT und BERT ist eine beträchtliche Sammlung (im Englischen als Model Zoo bezeichnet) von Transformer-Modellen entwickelt worden. Die chronologische Entwicklung mit den wichtigsten Fortschritten können Sie in Abbildung 1-1 nachvollziehen.

1 A. Vaswani et al., »Attention Is All You Need« (*https://arxiv.org/abs/1706.03762*), (2017). Dieser Titel war so einprägsam, dass nicht weniger als 50 Folgeartikel (*https://oreil.ly/wT8Ih*) »all you need« in ihren Titeln enthalten!

2 J. Howard und S. Ruder, »Universal Language Model Fine-Tuning for Text Classification« (*https://arxiv.org/abs/1801.06146*), (2018).

3 A. Radford et al., »Improving Language Understanding by Generative Pre-Training« (*https://openai.com/blog/language-unsupervised*), (2018).

4 J. Devlin et al., »BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding« (*https://arxiv.org/abs/1810.04805*), (2018).

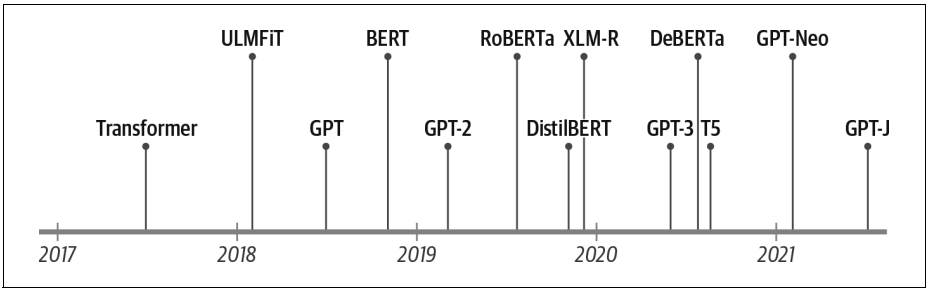

Abbildung 1-1: Chronologische Entwicklung von Transformer-Modellen

Doch greifen wir nicht zu weit vor. Um zu verstehen, was das Neue an Transformer-Modellen ist, müssen wir zunächst einige Begrifflichkeiten erläutern:

- das Encoder-Decoder-Framework
- den Attention-Mechanismus
- Transfer Learning

In diesem Kapitel stellen wir Ihnen die grundlegenden Konzepte vor, die für die weite Verbreitung von Transformern verantwortlich sind, und werfen einen Blick auf die Aufgaben, für die sich diese besonders eignen.

Lassen Sie uns zunächst das Encoder-Decoder-Framework und die Architekturen erkunden, die der Entwicklung von Transformer-Modellen vorausgingen.

Das Encoder-Decoder-Framework

Im Bereich des Natural Language Processing (NLP) galten vor den Transformern rekurrente Architekturen wie LSTMs als State of the Art. Diese Architekturen enthalten eine Rückkopplungsschleife in den Netzwerkverbindungen, die es ermöglicht, Informationen von einem Schritt zum nächsten weiterzugeben – dadurch eignen sie sich ideal für die Modellierung sequenzieller Daten wie Texte. Wie auf der linken Seite von Abbildung 1-2 dargestellt, empfängt ein RNN eine Eingabe bzw. einen Input (z.B. ein Wort oder ein Zeichen), leitet sie durch das Netzwerk und gibt einen Vektor aus, den sogenannten *Hidden State* bzw. *verborgenen Zustand* (auch verdeckter Zustand genannt). Gleichzeitig gibt das Modell über die Rückkopplungsschleife einige Informationen an sich selbst zurück, die es dann im nächsten Schritt verwenden kann. Das wird noch deutlicher, wenn wir die Schleife weiter »aufdröseln« bzw. in zeitlicher Hinsicht desaggregieren, wie auf der rechten Seite von Abbildung 1-2 gezeigt: Das RNN gibt in jedem Schritt Informationen über seinen Zustand an die nächste Operation in der Sequenz weiter. Auf diese Weise kann ein RNN die Informationen aus den vorangegangenen Schritten berücksichtigen und sie für seine Vorhersagen nutzen.

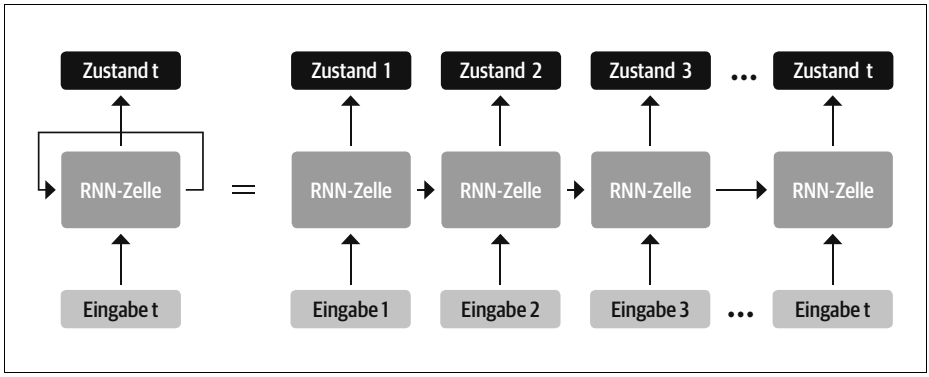

Abbildung 1-2: Zeitliche Desaggregation eines RNN

Diese Architekturen wurden (und werden) häufig für Aufgaben im Bereich des NLP, der Sprachverarbeitung (engl. Speech Processing) und für Zeitreihenanalysen verwendet. Eine wunderbare Übersicht, die Ihnen ein Bild davon vermittelt, wozu sie in der Lage sind, finden Sie in dem von Andrej Karpathy verfassten Blogbeitrag »The Unreasonable Effectiveness of Recurrent Neural Networks« (*https://oreil.ly/Q55o0*).

Ein Bereich, in dem RNNs eine wichtige Rolle spielten, war die Entwicklung von maschinellen Übersetzungssystemen. Bei diesen geht es darum, eine Folge von Wörtern aus einer Sprache in eine andere zu überführen. Diese Art von Aufgabe wird normalerweise mit einer *Encoder-Decoder-* bzw. einer *Sequence-to-Sequence-*Architektur[5] bewältigt, die sich gut für Situationen eignet, in denen sowohl die Eingabe (engl. Input) als auch die Ausgabe (engl. Output) Sequenzen beliebiger Länge darstellen. Die Aufgabe des Encoders ist es, die Informationen aus der Eingabesequenz in eine numerische Darstellung zu codieren, die oft als *letzter verborgener Zustand* (engl. Last Hidden State) bezeichnet wird. Dieser Zustand wird anschließend an den Decoder weitergegeben, der die Ausgabesequenz erzeugt.

Im Allgemeinen können die Encoder- und Decoder-Komponenten jede Art von neuronaler Netzwerkarchitektur sein, mit der sich Sequenzen modellieren lassen. In dem Beispiel in Abbildung Abbildung 1-3 werden z. B. mehrere RNNs verwendet, um für den englischen Satz »Transformers are great!« (der als verborgener Zustandsvektor codiert und dann decodiert wird) die deutsche Übersetzung »Transformer sind grossartig!« zu erzeugen. Die eingegebenen Wörter werden nacheinander durch den Encoder geleitet und die ausgegebenen Wörter – jeweils eines nach dem anderen, d. h. von oben nach unten – erzeugt.

5 I. Sutskever, O. Vinyals, and Q.V. Le, »Sequence to Sequence Learning with Neural Networks« (*https://arxiv.org/abs/1409.3215*), (2014).

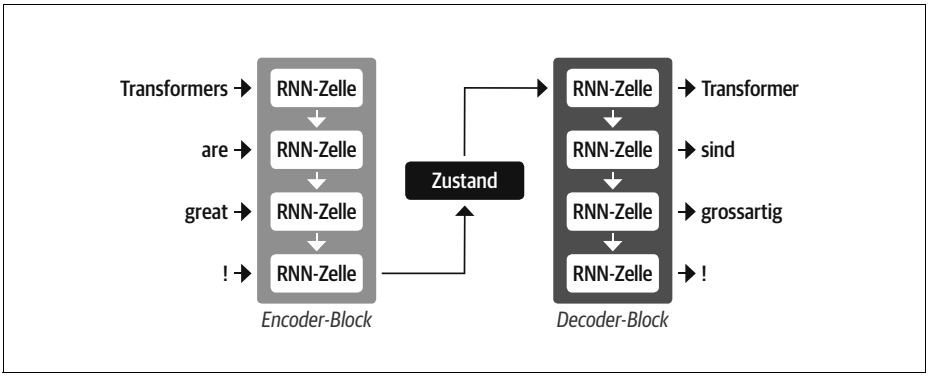

Abbildung 1-3: Eine Encoder-Decoder-Architektur mit mehreren RNNs (in der Regel gibt es bedeutend mehr rekurrente Schichten bzw. Layer als hier dargestellt)

Obwohl sie in ihrer Einfachheit elegant ist, besteht eine Schwäche dieser Architektur darin, dass der verborgene Endzustand des Encoders einen *Informationsengpass* darstellt: Er muss die Bedeutung der gesamten Eingabesequenz repräsentieren bzw. abbilden, da dies alles ist, worauf der Decoder bei der Erzeugung der Ausgabe zugreifen kann. Das ist vor allem bei langen Sequenzen eine Herausforderung, da Informationen, die am Anfang der Sequenz enthalten sind, bei der Komprimierung auf eine einzelne, starre Darstellung verloren gehen können.

Glücklicherweise gibt es einen Weg, diesen Engpass zu umgehen, indem der Decoder Zugriff auf alle verborgenen Zustände des Encoders erhält. Der grundlegende Ansatz für diese Lösung heißt *Attention*[6], der eine Schlüsselkomponente in vielen modernen neuronalen Netzwerkarchitekturen darstellt. Wenn wir nachvollziehen, wie der Attention-Mechanismus in RNNs implementiert wurde, können wir einen der wichtigsten Bausteine der Transformer-Architektur besser verstehen. Werfen wir einen genaueren Blick darauf.

Der Attention-Mechanismus

Die Hauptidee hinter Attention ist, dass der Encoder nicht nur einen einzigen verborgenen Zustand für die Eingabesequenz erzeugt, sondern bei jedem Schritt einen verborgenen Zustand (engl. Hidden State) ausgibt, auf den der Decoder zugreifen kann. Würden jedoch alle Zustände gleichzeitig verwendet, würde der Decoder eine sehr große Eingabe erhalten, sodass ein Mechanismus erforderlich ist, um Prioritäten bei der Verwendung der Zustände zu setzen. Hier kommt Attention ins Spiel: Der Decoder kann jedem der Encoderzustände bei jedem Decodierschritt eine andere Gewichtung zuweisen bzw. »Aufmerksamkeit« schenken. Dieser Prozess wird in Abbildung 1-4 veranschaulicht, wo gezeigt wird, welche Rolle die Attention für die Vorhersage des zweiten Tokens in der Ausgabesequenz einnimmt.

6 D. Bahdanau, K. Cho, and Y. Bengio, »Neural Machine Translation by Jointly Learning to Align and Translate« (*https://arxiv.org/abs/1409.0473*), (2014).

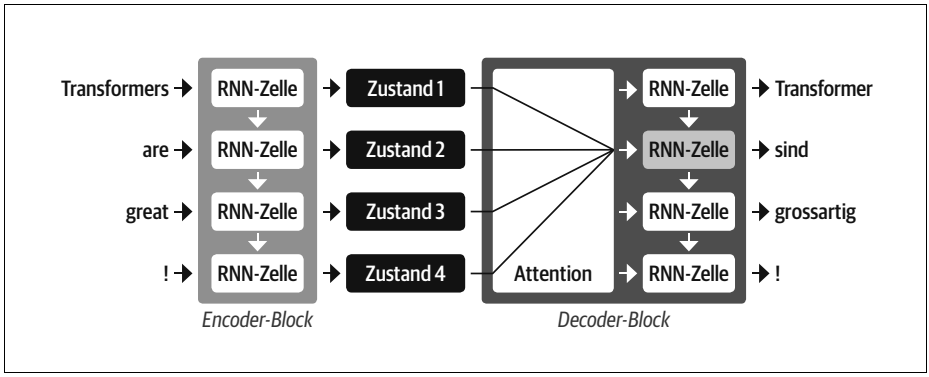

Abbildung 1-4: Eine Encoder-Decoder-Architektur mit Attention-Mechanismus für mehrere RNNs

Indem sie sich darauf konzentrieren, welche der eingegebenen Tokens zu jedem Zeitpunkt (also Schritt) am relevantesten sind, sind diese auf Attention basierenden Modelle in der Lage, nicht-triviale Zuordnungen zwischen den Wörtern in einer generierten Übersetzung und denen in einem Ausgangssatz zu lernen. In Abbildung 1-5 werden zum Beispiel die Attention-Gewichte für ein Modell, das Übersetzungen vom Englischen ins Französische vornimmt, dargestellt, wobei jedes Pixel einem Gewicht entspricht. Die Abbildung zeigt, wie der Decoder in der Lage ist, die Wörter »zone« und »Area«, die in den beiden Sprachen unterschiedlich angeordnet sind, korrekt zuzuordnen.

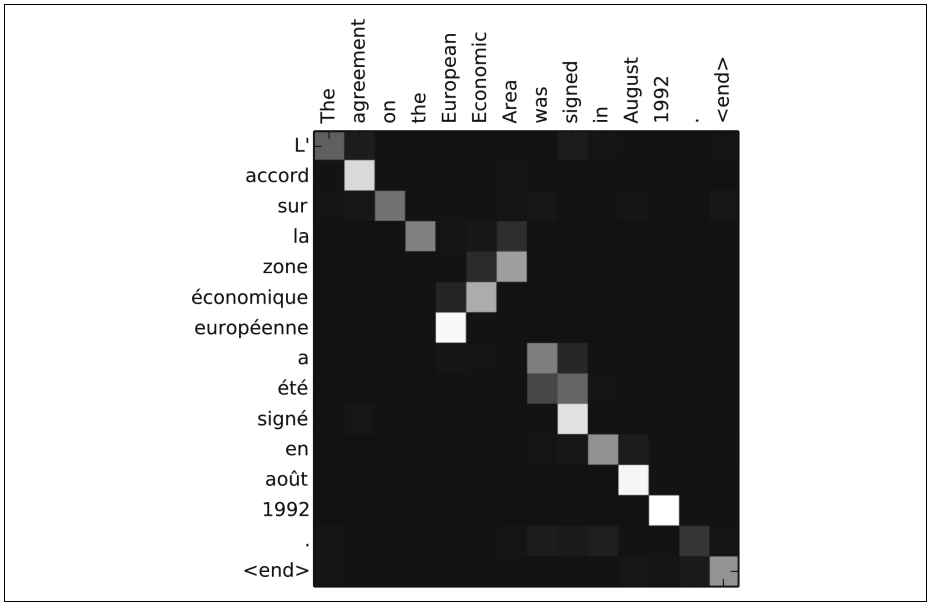

Abbildung 1-5: Zuordnung der Wörter zwischen einem englischen Ausgangstext und der generierten Übersetzung ins Französische eines Encoder-Decoder-Modells auf Basis von RNNs (mit freundlicher Genehmigung von Dzmitry Bahdanau)

Obwohl infolge der Verwendung von Attention bedeutend bessere Übersetzungen erzielt werden konnten, gab es bei der Verwendung von rekurrenten Modellen als Encoder und Decoder einen großen Nachteil: Die Berechnungen sind von Natur aus sequenziell und können nicht für die gesamte Eingabesequenz parallelisiert werden.

Mit dem Transformer wurde ein neues Modellierungsparadigma eingeführt: Bei dieser Architektur wird auf die Rekursion verzichtet und stattdessen vollständig auf eine besondere Form von Attention, der sogenannten *Self-Attention*, zurückgegriffen. Was Self-Attention genau ist, werden wir noch in Kapitel 3 erläutern. Die Grundidee hierbei ist, dass die Attention auf alle Zustände *derselben Schicht* des neuronalen Netzes operieren kann. Dies können Sie in Abbildung 1-6 nachvollziehen, wo sowohl der Encoder als auch der Decoder ihre eigenen Self-Attention-Mechanismen haben, deren Ausgaben in neuronale Feed-Forward-Netze (engl. Feed-Forward Neural Networks) eingespeist werden. Diese Architektur kann bedeutend schneller trainiert werden als rekurrente Modelle und ebnete vielen der jüngsten Durchbrüche im NLP den Weg.

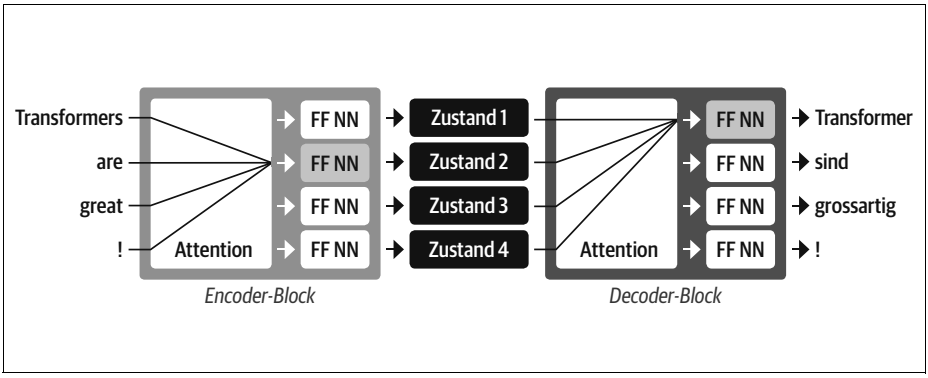

Abbildung 1-6: Encoder-Decoder-Architektur des ursprünglichen Transformers

In dem ursprünglichen Artikel zu Transformer-Modellen wurde das Übersetzungsmodell anhand eines großen Korpus von Satzpaaren, die in verschiedenen Sprachen vorliegen, von Grund auf trainiert. In vielen Fällen, in denen NLP in der Praxis zum Einsatz kommt, gibt es jedoch keine große Menge an gelabelten Textdaten, mit denen wir unsere Modelle trainieren könnten. Um die Transformer-Revolution vollends in Gang zu setzen, fehlte daher noch ein letzter Baustein: Transfer Learning.

Einsatz von Transfer Learning im NLP

Im Bereich der Computer Vision (computerbasiertes Sehen) ist es heutzutage üblich, ein Convolutional Neural Network (CNN, auch neuronales Konvolutionsnetz genannt) wie ResNet mithilfe von Transfer Learning für eine Aufgabe zu trainieren und es dann an eine neue Aufgabe anzupassen bzw. es für diese feinzutunen (engl.

fine-tune). Auf diese Weise kann das Netz das Wissen nutzen, das es bei der ursprünglichen Aufgabe gelernt hat. Architektonisch bedeutet dies, dass das Modell in einen *Body* (bzw. Körper) und einen *Head* (bzw. Kopf) aufgeteilt wird, wobei der Head ein aufgabenspezifisches Netz darstellt. Während des Trainings lernt der Body mittels seiner Gewichte allgemeine Merkmale (engl. Features) der Ausgangsdomäne. Diese Gewichte werden verwendet, um ein neues Modell, das für die neue Aufgabe bestimmt ist, zu initialisieren.[7] Im Vergleich zum traditionellen überwachten Lernen (engl. Supervised Learning) führt dieser Ansatz in der Regel zu qualitativ hochwertigen Modellen, die bedeutend effizienter für eine Vielzahl von nachgelagerten Aufgaben (engl. Downstream Tasks) und mit weit weniger gelabelten Daten trainiert werden können. Einen Vergleich der beiden Ansätze finden Sie in Abbildung 1-7.

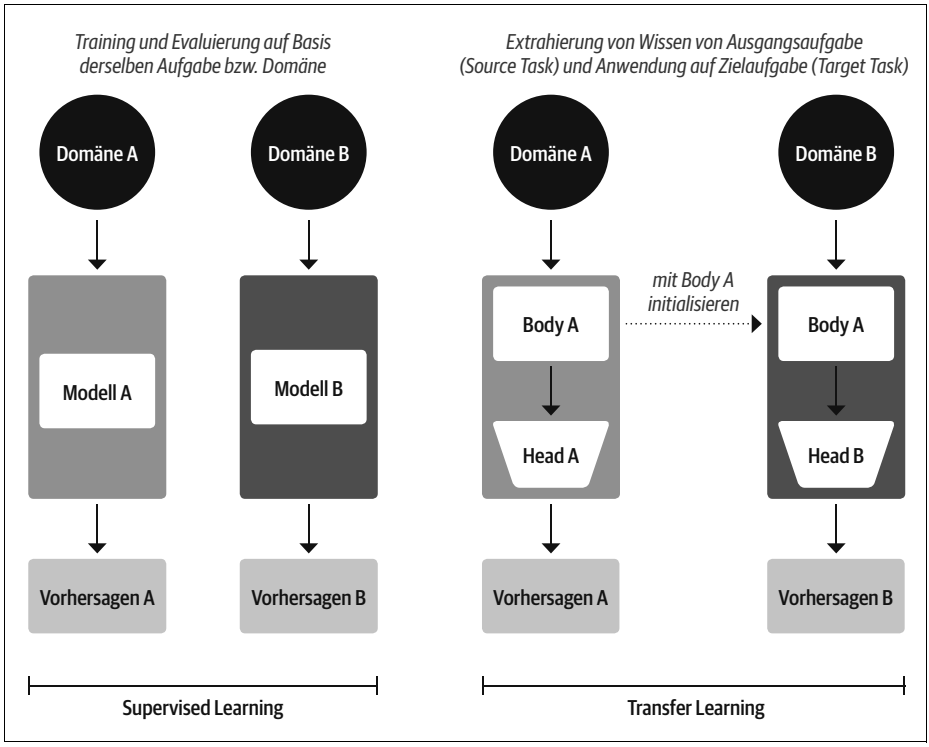

Abbildung 1-7: Vergleich von traditionellem Supervised Learning (links) und Transfer Learning (rechts)

In der Computer Vision werden die Modelle zunächst auf großen Datensätzen wie ImageNet (*https://image-net.org*) trainiert, die Millionen von Bildern enthalten. Dieser Vorgang wird als *Pretraining* (bzw. Vortraining) bezeichnet und dient vor allem dazu, den Modellen die grundlegenden Merkmale von Bildern, wie Kanten oder Farben, beizubringen. Diese vortrainierten Modelle können dann für eine

7 Gewichte sind die lernbaren Parameter eines neuronalen Netzes.

nachgelagerte Aufgabe, wie z. B. die Klassifizierung von Blumenarten, mit einer relativ kleinen Anzahl von gelabelten Beispielen (in der Regel einige Hundert pro Kategorie) feingetunt werden. Solche (im Rahmen des Feintunings) optimierten Modelle erreichen in der Regel eine höhere Treffergenauigkeit (engl. Accuracy) als überwachte Modelle, die mit der gleichen Menge an gelabelten Daten von Grund auf trainiert wurden.

Obwohl sich das Transfer Learning in der Computer Vision durchgesetzt hat, war lange Zeit nicht klar, wie das entsprechende Pretraining beim NLP gestaltet sein sollte. Dementsprechend benötigten NLP-Anwendungen in der Regel große Mengen an gelabelten Daten, um eine hohe Leistungsfähigkeit zu erreichen. Allerdings war selbst dann die Leistung nicht vergleichbar mit dem, was im Bereich Computer Vision erreicht werden konnte.

In den Jahren 2017 und 2018 wurden neue Ansätze von verschiedenen Forschungsgruppen vorgeschlagen, die schließlich dazu führten, dass Transfer Learning auch für das NLP nutzbar gemacht werden konnte. Es begann damit, dass Forscher bei OpenAI erkannten, dass eine starke Leistung bei einer Sentiment-Klassifizierungsaufgabe erzielt wurde, indem sie Features verwendeten, die im Rahmen eines unüberwachten Pretrainings gewonnen wurden.[8] Im Anschluss folgte ULMFiT, mit dem ein allgemeiner Rahmen für die Verwendung vortrainierter LSTM-Modelle für verschiedene Aufgaben geschaffen wurde.[9]

Wie in Abbildung 1-8 dargestellt, umfasst ULMFiT drei Hauptschritte:

Pretraining
: Das anfängliche Ziel des Trainings ist ganz einfach: das nächste Wort auf der Grundlage der vorherigen Wörter vorherzusagen. Diese Aufgabe wird als *Sprachmodellierung* (engl. Language Modeling) bezeichnet. Die Eleganz dieses Ansatzes liegt darin, dass keine gelabelten Daten benötigt werden und auf reichlich vorhandenen Text aus Quellen wie Wikipedia zurückgegriffen werden kann.[10]

Domänenadaption (engl. Domain Adaptation)
: Sobald das Sprachmodell auf ein großes Korpus vortrainiert wurde, wird es im nächsten Schritt auf ein domänenspezifisches Korpus übertragen (z. B. von Wikipedia auf das IMDb-Korpus, das Filmrezensionen enthält, wie in Abbildung 1-8 dargestellt). Zwar wird auch in diesem Schritt auf die Sprachmodellierung zurückgegriffen, allerdings muss das Modell in diesem Zusammenhang das nächste Wort im Zielkorpus vorhersagen.

8 A. Radford, R. Jozefowicz und I. Sutskever, »Learning to Generate Reviews and Discovering Sentiment« (*https://arxiv.org/abs/1704.01444*), (2017).

9 Eine verwandte Veröffentlichung zu dieser Zeit war ELMo (Embeddings from Language Models), mit der gezeigt wurde, wie durch ein Pretraining von LSTMs qualitativ hochwertige Worteinbettungen – aus dem Englischen auch häufig als Word Embeddings bezeichnet – für nachgelagerte Aufgaben erzeugt werden können.

10 Dies gilt eher für das Englische als für die meisten anderen Sprachen der Welt, für die es schwierig sein kann, ein großes Korpus digitalisierter Texte zu erhalten. Die Suche nach Möglichkeiten, diese Lücke zu schließen, ist ein aktiver Bereich der NLP-Forschung und -Bemühungen.

Feintuning (engl. Fine-tuning)
: In diesem Schritt wird das Sprachmodell mit einer (zusätzlichen) Klassifizierungsschicht bzw. einem Klassifizierungslayer für die Zielaufgabe (engl. Target Task) feingetunt (z.B. zur Klassifizierung des Sentiments bzw. der Polarität von Filmrezensionen in Abbildung 1-8).

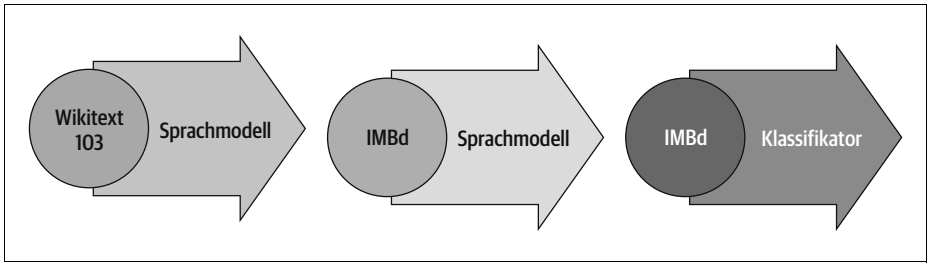

Abbildung 1-8: Das Vorgehen bei ULMFiT (mit freundlicher Genehmigung von Jeremy Howard)

Die Einführung von ULMFiT bot somit einen praktikablen Rahmen für das Pretraining und die Anwendung von Transfer Learning im Bereich des NLP und lieferte damit das fehlende Puzzleteil, um Transformern zum Durchbruch zu verhelfen. Im Jahr 2018 wurden zwei Transformer-Modelle veröffentlicht, bei denen Self-Attention mit Transfer Learning kombiniert wurde:

GPT
: Verwendet nur den Decoder-Teil der Transformer-Architektur und denselben Sprachmodellierungsansatz wie ULMFiT. GPT wurde mithilfe des BookCorpus-Datensatzes[11] vortrainiert, der aus 7.000 unveröffentlichten Büchern aus verschiedenen Genres wie Abenteuer, Fantasy und Romantik besteht.

BERT
: Verwendet den Encoder-Teil der Transformer-Architektur und eine spezielle Form der Sprachmodellierung, die als *Masked Language Modeling* bezeichnet wird. Das Ziel dieses Ansatzes besteht darin, in einem Text Wörter vorherzusagen, die zufällig maskiert bzw. verdeckt wurden. Bei einem Satz wie »Ich habe auf meine `[MASK]` geschaut und gesehen, dass sich `[MASK]` verspätet hat.« muss das Modell jeweils die wahrscheinlichsten Kandidaten für die maskierten Wörter, die durch `[MASK]` gekennzeichnet sind, vorhersagen. BERT wurde auf dem BookCorpus-Datensatz und der englischsprachigen Wikipedia vortrainiert.

Sowohl GPT als auch BERT haben den Stand der Technik (State of the Art) in einer Reihe von NLP-Benchmarks neu gesetzt und das Zeitalter der Transformer-Modelle eingeläutet.

Da jedoch verschiedene Forschungseinrichtungen ihre Modelle in inkompatiblen Frameworks (PyTorch oder TensorFlow) veröffentlichten, war es für NLP-Praktike-

11 Y. Zhu et al., »Aligning Books and Movies: Towards Story-Like Visual Explanations by Watching Movies and Reading Books« (*https://arxiv.org/abs/1506.06724*), (2015).

rinnen und -Praktiker nicht immer einfach, diese Modelle auf ihre eigenen Anwendungen zu übertragen. Mit der Veröffentlichung der 🤗 Transformers-Bibliothek (*https://oreil.ly/Z79jF*) wurde nach und nach eine einheitliche Programmierschnittstelle (API) für mehr als 50 Architekturen entwickelt. Dank dieser Bibliothek wurde die Forschung im Bereich von Transformer-Modellen explosionsartig vorangetrieben, und die Praktikerinnen und Praktiker im Bereich NLP konnten diese Modelle innerhalb kurzer Zeit auf einfache Weise in viele reale Anwendungen integrieren. Lassen Sie uns einen Blick darauf werfen!

Die Transformers-Bibliothek von Hugging Face: die Lücke schließen

Eine neue Machine-Learning-Architektur auf eine neue Aufgabe anzuwenden, kann ein komplexes Unterfangen sein und umfasst in der Regel die folgenden Schritte:

1. Die Modellarchitektur als Code zu implementieren, typischerweise auf Basis von PyTorch oder TensorFlow.
2. Die vortrainierten Gewichte (falls verfügbar) von einem Server zu laden.
3. Die Eingaben (engl. Inputs) vorzuverarbeiten (engl. Preprocessing), sie durch das Modell laufen zu lassen und je nach Aufgabe, die verfolgt wird, nachzuverarbeiten (engl. Postprocessing).
4. Die Objekte, mit denen sich die Daten laden lassen, sogenannte Dataloader, zu implementieren und die Verlustfunktionen und den Optimizer bzw. den Optimierungsalgorithmus, die zum Trainieren des Modells verwendet werden, zu bestimmen.

Je nach Modell und Aufgabe sind jeweils verschiedenartige Schritte erforderlich. Wenn Forschungsgruppen einen neuen Artikel veröffentlichen, geben sie üblicherweise (aber nicht immer!) auch den Code zusammen mit der Modellgewichtung frei. Der Code ist jedoch selten standardisiert, und es erfordert oft tagelange Entwicklungsarbeit, ehe er so angepasst ist, dass er für neue Anwendungsfälle verwendet werden kann.

Genau an diesem Punkt kommt die Transformers-Bibliothek den NLP-Anwenderinnen und -Anwendern zu Hilfe! Sie bietet eine standardisierte Schnittstelle zu einer breiten Palette von Transformer-Modellen sowie Code und Tools, mit denen sich die Modelle auf neue Anwendungsfälle übertragen lassen. Die Bibliothek unterstützt derzeit drei wichtige Deep-Learning-Frameworks (PyTorch, TensorFlow und JAX) und ermöglicht es Ihnen, ohne Weiteres zwischen ihnen zu wechseln. Darüber hinaus bietet sie aufgabenspezifische Heads, sodass Sie Transformer für nachgelagerte Aufgaben wie die Textklassifizierung, die Eigennamenerkennung (engl. Named Entity Recognition, NER) oder die Beantwortung von Fragen (engl. Question Answering) problemlos feintunen können.[12] Dadurch verkürzt sich die

12 Anm. d. Übersetzers: Um eine bessere Nachvollziehbarkeit und Übertragbarkeit in die Praxis zu gewährleisten, verwenden wir in diesem Buch vornehmlich die englischen Begriffe.

Zeit, die Sie zum Trainieren und Testen einer Handvoll von Modellen benötigen, von einer Woche auf einen einzigen Nachmittag!

Im nächsten Abschnitt werden wir Ihnen zeigen, dass Sie einige der häufigsten NLP-Anwendungsfälle, die Ihnen in der Praxis begegnen werden, mithilfe der Transformers-Bibliothek mit nur wenigen Zeilen Code angehen können.

Die Anwendungsmöglichkeiten von Transformern im Überblick

Am Anfang einer jeden NLP-Aufgabe liegt ein Text vor, wie z. B. das folgende, frei erfundene Kundenfeedback zu einer bestimmten Onlinebestellung:

```
text = """Dear Amazon, last week I ordered an Optimus Prime action figure
from your online store in Germany. Unfortunately, when I opened the package,
I discovered to my horror that I had been sent an action figure of Megatron
instead! As a lifelong enemy of the Decepticons, I hope you can understand my
dilemma. To resolve the issue, I demand an exchange of Megatron for the
Optimus Prime figure I ordered. Enclosed are copies of my records concerning
this purchase. I expect to hear from you soon. Sincerely, Bumblebee."""
```

Je nach Ihrer Anwendung kann der Text, mit dem Sie arbeiten, ein juristischer Vertrag, eine Produktbeschreibung oder etwas ganz anderes sein. Im Falle von Kundenfeedback möchten Sie wahrscheinlich wissen, ob das Feedback positiv oder negativ ist. Diese Aufgabe wird *Sentimentanalyse* bzw. Stimmungsanalyse genannt und ist Teil des umfassenderen Bereichs der *Textklassifizierung* (engl. Text Classification), den wir in Kapitel 2 beleuchten werden. Schauen wir uns zunächst einmal an, was nötig ist, um die Stimmungslage bzw. das Sentiment unseres Texts mithilfe der 🤗 Transformers-Bibliothek zu ermitteln.

Textklassifizierung

Wie wir in den folgenden Kapiteln sehen werden, verfügt die 🤗 Transformers-Bibliothek über eine abgestufte API, die es Ihnen ermöglicht, mit der Bibliothek auf verschiedenen Abstraktionsebenen zu interagieren. In diesem Kapitel beginnen wir mit *Pipelines*, mit denen alle Schritte abstrahiert werden, die notwendig sind, um einen Rohtext in eine Reihe von Vorhersagen auf Basis eines feingetunten Modells umzuwandeln.

In der 🤗 Transformers-Bibliothek instanziieren wir eine Pipeline, indem wir die Funktion `pipeline()` aufrufen und den Namen der Aufgabe angeben, die für uns von Interesse ist:

```
from transformers import pipeline

classifier = pipeline("text-classification")
```

Wenn Sie diesen Code zum ersten Mal ausführen, werden einige Fortschrittsbalken angezeigt, da die Pipeline die Modellgewichtung automatisch vom Hugging

Face Hub (*https://oreil.ly/zLK11*) herunterlädt. Wenn Sie die Pipeline ein zweites Mal instanziieren, stellt die Bibliothek fest, dass Sie die Gewichtung bereits heruntergeladen haben, und verwendet stattdessen die im Cache gespeicherte Version. Die `text-classification`-Pipeline verwendet standardmäßig ein Modell, das für die Sentimentanalyse entwickelt wurde, unterstützt aber auch Multiklassen- (bzw. mehrkategoriale) und Multilabel-Klassifizierung.

Nachdem wir unsere Pipeline erstellt haben, können wir nun einige Vorhersagen treffen! Jede Pipeline nimmt eine Textzeichenkette bzw. Strings (oder eine Liste von Strings) als Eingabe entgegen und gibt eine Liste von Vorhersagen zurück. Jede Vorhersage entspricht einem Python-Dictionary, weshalb wir auf Pandas zurückgreifen können, um sie als `DataFrame` darzustellen:

```
import pandas as pd

outputs = classifier(text)
pd.DataFrame(outputs)
```

	Label	Score
0	NEGATIVE	0.901546

In diesem Fall geht das Modell mit hoher Wahrscheinlichkeit davon aus, dass der Text ein negatives Stimmungsbild vermittelt. Wenn Sie sich vor Augen führen, dass es sich um eine Beschwerde eines verärgerten Kunden handelt, erscheint dies durchaus eine plausible Einschätzung zu sein! Beachten Sie, dass die Pipeline im Rahmen von Aufgaben der Sentimentanalyse nur eines der beiden verwendeten Labels, `POSITIVE` oder `NEGATIVE`, zurückgibt, da beide Scores bzw. Werte in Summe eins ergeben und der jeweils andere Wert dementsprechend durch die Berechnung von `1-Score` ermittelt werden kann.

Werfen wir nun einen Blick auf eine andere häufig anzutreffende Aufgabenstellung: die Identifizierung von Eigennamen bzw. benannten Entitäten in Texten.

Named Entity Recognition

Die Vorhersage des Stimmungsbilds des Kundenfeedbacks ist ein guter erster Schritt. Oft möchten Sie jedoch erfahren, ob sich das Feedback auf einen bestimmten Artikel oder eine bestimmte Dienstleistung bezieht. Im NLP werden Objekte der realen Welt wie Produkte, Orte und Personen als *Named Entities*, also Eigennamen bzw. benannte Entitäten, bezeichnet, und das Extrahieren dieser Objekte aus Text wird *Eigennamenerkennung* bzw. *Named Entity Recognition* (NER) genannt. Um eine NER durchführen zu können, müssen wir die entsprechende Pipeline laden und sie mit unserer Kundenrezension füttern:

```
ner_tagger = pipeline("ner", aggregation_strategy="simple")
outputs = ner_tagger(text)
pd.DataFrame(outputs)
```

	entity_group	score	word	start	end
0	ORG	0.879010	Amazon	5	11
1	MISC	0.990859	Optimus Prime	36	49
2	LOC	0.999755	Germany	90	97
3	MISC	0.556569	Mega	208	212
4	PER	0.590256	##tron	212	216
5	ORG	0.669692	Decept	253	259
6	MISC	0.498350	##icons	259	264
7	MISC	0.775361	Megatron	350	358
8	MISC	0.987854	Optimus Prime	367	380
9	PER	0.812096	Bumblebee	502	511

Wie Sie sehen, hat die Pipeline alle Entitäten erkannt und jeder von ihnen eine Kategorie wie `ORG` (Organisation), `LOC` (Ort, engl. Location) oder `PER` (Person) zugewiesen. Hier haben wir das Argument `aggregation_strategy` verwendet, um die Wörter entsprechend den Vorhersagen des Modells zu gruppieren. Die Entität »Optimus Prime« besteht zum Beispiel aus zwei Wörtern, wird aber einer einzigen Kategorie zugeordnet: `MISC` (Verschiedenes, engl. Miscellaneous). Anhand der Scores können wir beurteilen, mit welcher Konfidenz das Modell die identifizierten Entitäten bzw. Eigennamen einschätzt. Wir sehen, dass es bei »Decepticons« und dem ersten Vorkommen von »Megatron« am unsichersten war, da es beide nicht als eine einzelne Entität zusammenfassen konnte.

Sehen Sie in der vorherigen Tabelle diese seltsamen Hash-Symbole (`#`) in der Spalte `word`? Diese werden vom *Tokenizer* des Modells erzeugt, der Wörter in einzelne Einheiten, sogenannte *Tokens*, zerlegt. In Kapitel 2 werden Sie noch mehr über die Tokenisierung erfahren.

Alle Eigennamen (bzw. benannte Entitäten) in einem Text zu identifizieren, ist hilfreich, doch manchmal würden wir gerne gezieltere Fragen stellen. Hierfür können wir auf das *Question Answering* (QA) bzw. die *automatische Fragenbeantwortung* zurückgreifen.

Question Answering

Beim Question Answering übermitteln wir dem Modell eine Textpassage, die wir als *Kontext* (engl. Context) bezeichnen, zusammen mit einer Frage, deren Antwort wir gerne extrahieren möchten. Das Modell gibt dann die Passage des Texts zurück, die der Antwort entspricht. Sehen wir uns einmal an, was wir erhalten, wenn wir eine konkrete Frage bezüglich unseres Kundenfeedbacks stellen:

```
reader = pipeline("question-answering")
question = "What does the customer want?"
outputs = reader(question=question, context=text)
pd.DataFrame([outputs])
```

	score	start	end	answer
0	0.631291	335	358	an exchange of Megatron

Wie wir sehen, hat die Pipeline neben der Antwort auch die Ganzzahlen `start` und `end` zurückgegeben, die den Indexen der Zeichen entsprechen, in denen die Antwort gefunden wurde (genau wie beim NER-Tagging). Dieser eingegrenzte Bereich wird als Antwortspanne (engl. Answer Span) bezeichnet, die den Beginn und das Ende der Textpassage, in der die Antwort enthalten ist, kennzeichnet. Es gibt verschiedene Arten des Question Answering, auf die wir noch in Kapitel 7 eingehen werden. Diese spezielle Art wird als *extraktives Question Answering* bezeichnet, da die Antwort direkt aus dem Text extrahiert wird.

Mit diesem Ansatz können Sie schnell relevante Informationen aus dem Feedback eines Kunden erkennen und extrahieren. Aber was ist, wenn Sie einen Berg von langatmigen Beschwerden erhalten und nicht die Zeit haben, sie alle zu lesen? Mal sehen, ob ein Modell, das Zusammenfassungen generiert, helfen kann!

Automatische Textzusammenfassung (Summarization)

Das Ziel der automatischen Textzusammenfassung (engl. Summarization) ist es, aus einem langen Text eine Kurzfassung zu erstellen, die alle relevanten Fakten enthält. Dies ist eine wesentlich kompliziertere Aufgabe als die vorherigen, da das Modell einen zusammenhängenden Text *generieren* muss. Wie Sie es bereits von den vorherigen Aufgaben kennen, können wir eine Zusammenfassungspipeline wie folgt instanziieren:

```
summarizer = pipeline("summarization")
outputs = summarizer(text, max_length=45, clean_up_tokenization_spaces=True)
print(outputs[0]['summary_text'])
```

```
 Bumblebee ordered an Optimus Prime action figure from your online store in German
y. Unfortunately, when I opened the package, I discovered to my horror that I had
been sent an action figure of Megatron instead.
```

Die ausgegebene Zusammenfassung ist gar nicht mal so schlecht! Obwohl Teile des ursprünglichen Texts lediglich kopiert wurden, war das Modell in der Lage, das Wesentliche des Problems zu erfassen und korrekt zu erkennen, dass »Bumblebee« (der am Ende des Texts vorkam) der Verfasser der Beschwerde ist. In diesem Beispiel können Sie auch sehen, dass wir einige Schlüsselwortargumente wie `max_length` und `clean_up_tokenization_spaces` an die Pipeline übergeben haben. Diese ermöglichen es uns, die Ergebnisse weiter zu optimieren.

Was aber, wenn Sie Feedback in einer Sprache erhalten, die Sie nicht verstehen? Dann können Sie Google Translate verwenden oder auch Ihr eigenes Transformer-Modell, das Ihnen die Übersetzung abnimmt!

Maschinelle Übersetzung (Translation)

Wie die automatische Textzusammenfassung ist auch die maschinelle Übersetzung (engl. Translation) eine Aufgabe, bei der die Ausgabe einem generierten Text entspricht. Verwenden wir nun eine Übersetzungspipeline, um einen englischen Text ins Deutsche zu übersetzen:

```
translator = pipeline("translation_en_to_de",
                      model="Helsinki-NLP/opus-mt-en-de")
outputs = translator(text, clean_up_tokenization_spaces=True, min_length=100)
print(outputs[0]['translation_text'])
```

```
Sehr geehrter Amazon, letzte Woche habe ich eine Optimus Prime Action Figur aus
Ihrem Online-Shop in Deutschland bestellt. Leider, als ich das Paket öffnete,
entdeckte ich zu meinem Entsetzen, dass ich stattdessen eine Action Figur von
Megatron geschickt worden war! Als lebenslanger Feind der Decepticons, Ich
hoffe, Sie können mein Dilemma verstehen. Um das Problem zu lösen, Ich fordere
einen Austausch von Megatron für die Optimus Prime Figur habe ich bestellt.
Anbei sind Kopien meiner Aufzeichnungen über diesen Kauf. Ich erwarte, bald von
Ihnen zu hören. Aufrichtig, Bumblebee.
```

Das Modell hat auch hier eine sehr gute Übersetzung hervorgebracht, bei der die förmlichen Pronomen der deutschen Sprache – wie »Ihrem« und »Sie« – korrekt verwendet werden. Außerdem haben wir hierdurch gezeigt, wie Sie das standardmäßig vorgegebene (engl. default) Modell in der Pipeline ersetzen, um das für Ihre Anwendung am besten geeignete Modell auswählen zu können – und auf dem Hugging Face Hub können Sie Modelle für Tausende von Sprachpaaren finden. Nehmen wir noch eine letzte Anwendung unter die Lupe, bevor wir wieder einen Schritt zurücktreten, um uns das gesamte Ökosystem von Hugging Face anzusehen.

Textgenerierung

Nehmen wir an, Sie möchten in der Lage sein, schneller auf Kundenfeedback zu antworten und dazu auf eine Autovervollständigungsfunktion zurückgreifen. Mit einem Textgenerierungsmodell (engl. Text Generation Model) könnten Sie dies wie folgt umsetzen:

```
generator = pipeline("text-generation")
response = "Dear Bumblebee, I am sorry to hear that your order was mixed up."
prompt = text + "\n\nCustomer service response:\n" + response
outputs = generator(prompt, max_length=200)
print(outputs[0]['generated_text'])
```

```
Dear Amazon, last week I ordered an Optimus Prime action figure from your online
store in Germany. Unfortunately, when I opened the package, I discovered to my
horror that I had been sent an action figure of Megatron instead! As a lifelong
enemy of the Decepticons, I hope you can understand my dilemma. To resolve the
issue, I demand an exchange of Megatron for the Optimus Prime figure I ordered.
Enclosed are copies of my records concerning this purchase. I expect to hear
from you soon. Sincerely, Bumblebee.
```

```
Customer service response:
Dear Bumblebee, I am sorry to hear that your order was mixed up. The order was
completely mislabeled, which is very common in our online store, but I can
appreciate it because it was my understanding from this site and our customer
service of the previous day that your order was not made correct in our mind and
that we are in a process of resolving this matter. We can assure you that your
order
```

Nun gut. Die generierte Vervollständigung würden wir wahrscheinlich nicht dazu verwenden, Bumblebee zu beruhigen. Die Idee dahinter sollte jedoch klar geworden sein.

Nachdem Sie nun ein paar spannende Anwendungen von Transformer-Modellen kennengelernt haben, fragen Sie sich vielleicht, wo diese trainiert werden. Alle Modelle, die wir in diesem Kapitel verwendet haben, sind frei verfügbar und bereits für die jeweilige Aufgabe feingetunt. Normalerweise möchten Sie die Modelle jedoch für Ihre eigenen Daten feintunen. In den folgenden Kapiteln werden Sie lernen, wie Sie das anstellen können.

Doch ein Modell zu trainieren, macht nur einen kleinen Teil eines jeden NLP-Projekts aus. Die Daten effizient zu verarbeiten, Ergebnisse mit Kollegen zu teilen und Ihre Arbeit reproduzierbar zu machen, sind ebenfalls wichtige Komponenten. Glücklicherweise ist die 🤗 Transformers-Bibliothek von einem großen Ökosystem nützlicher Tools umgeben, die einen Großteil des modernen Machine-Learning-Workflows unterstützen. Sehen wir uns dieses Ökosystem etwas genauer an.

Das Ökosystem von Hugging Face

Was mit der 🤗 Transformers-Bibliothek begann, hat sich schnell zu einem ganzen Ökosystem entwickelt, das aus vielen Bibliotheken und Tools besteht, mit denen Sie Ihre NLP- und Machine-Learning-Projekte schneller umsetzen können. Das Ökosystem von Hugging Face besteht hauptsächlich aus zwei Teilen: einer ganzen Reihe von Bibliotheken und dem Hub, wie in Abbildung 1-9 gezeigt. Die Bibliotheken stellen den Code zur Verfügung, während der Hub die Pretraining-Modellgewichte, Datensätze, Skripte für die Berechnung der Maße zur Evaluierung und mehr bereitstellt. In diesem Abschnitt werfen wir einen kurzen Blick auf die verschiedenen Komponenten. Wir überspringen die 🤗 Transformers-Bibliothek, da wir sie bereits vorgestellt haben und im Laufe des Buchs noch sehr viel mehr über sie erfahren werden.

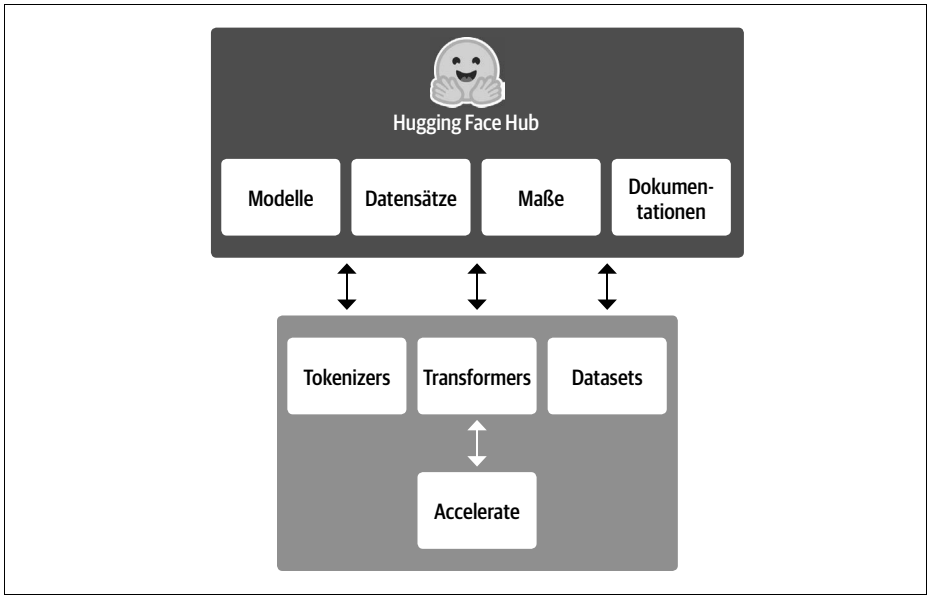

Abbildung 1-9: Ein Überblick über das Ökosystem von Hugging Face

Der Hugging Face Hub

Wie bereits erwähnt, ist Transfer Learning einer der Schlüsselfaktoren für den Erfolg von Transformer-Modellen, da es dadurch möglich ist, vortrainierte Modelle für neue Aufgaben wiederzuverwenden. Daher ist es von entscheidender Bedeutung, dass Sie vortrainierte Modelle schnell laden und Experimente mit ihnen durchführen können.

Der Hugging Face Hub enthält über 20.000 frei verfügbare Modelle. Wie Sie in Abbildung 1-10 sehen, gibt es Filter für Aufgaben (»Tasks«), Frameworks, Datensätze (»Datasets«) und vieles mehr. Diese sollen Ihnen helfen, sich im Hub zurechtzufinden und schnell vielversprechende Modellkandidaten zu finden. Wie Sie bereits bei der Verwendung von Pipelines feststellen konnten, ist das Laden eines vielversprechenden Modells dann buchstäblich nur eine Zeile Code entfernt. Das erleichtert es Ihnen, mit einer Vielzahl von Modellen zu experimentieren, und gibt Ihnen die Möglichkeit, sich auf die domänenspezifischen Aspekte Ihres Projekts zu konzentrieren.

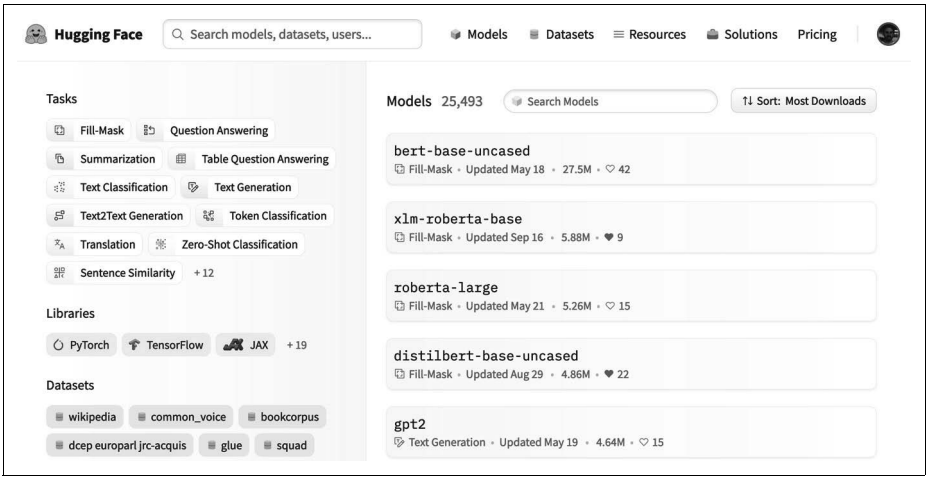

Abbildung 1-10: Die »Models«-Seite des Hugging Face Hub, mit Filterfunktionen auf der linken Seite und einer Liste von Modellen auf der rechten Seite

Zusätzlich zu den Modellgewichtungen bzw. -gewichten enthält der Hub auch Datensätze und Skripte zur Berechnung der Maße zur Evaluierung, mit denen Sie die veröffentlichten Ergebnisse reproduzieren oder zusätzliche Daten für Ihre Anwendung nutzen können.

Der Hub bietet auch sogenannte *Model-Cards* und *Dataset-Cards*, in denen allgemeine Aspekte zu den Modellen und Datensätzen dokumentiert sind und die Ihnen eine fundierte Entscheidung darüber ermöglichen sollen, ob sie die richtige Wahl für Sie darstellen. Eines der besten Features des Hub ist, dass Sie jedes Modell direkt über die verschiedenen aufgabenspezifischen interaktiven Widgets ausprobieren können (siehe Abbildung 1-11).

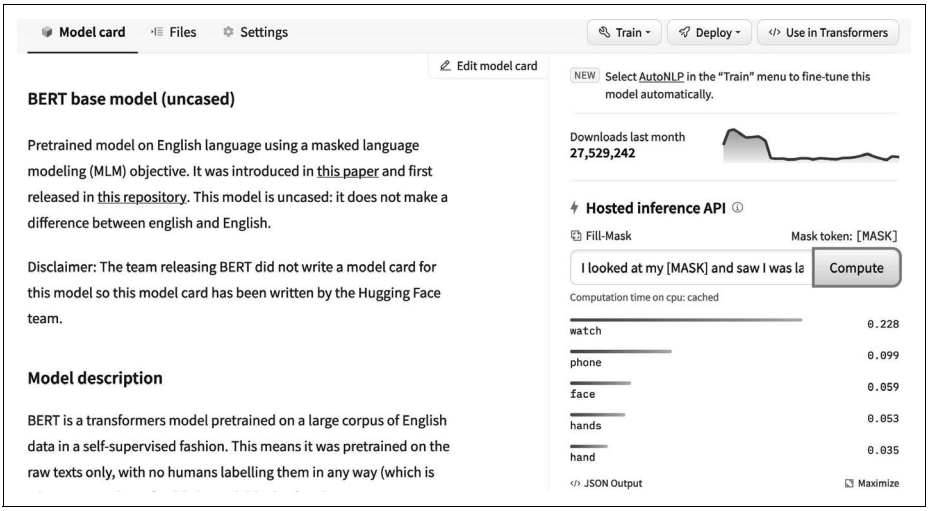

Abbildung 1-11: Ein Beispiel für eine Model-Card aus dem Hugging Face Hub: Das Inference-Widget, mit dem Sie mit dem Modell interagieren können, sehen Sie rechts.

Kommen wir nun zur 🤗 Tokenizers-Bibliothek.

PyTorch (*https://oreil.ly/AyTYC*) und TensorFlow (*https://oreil.ly/JOKgq*) bieten auch eigene Hubs, deren Nutzung sich insbesondere dann anbietet, wenn ein bestimmtes Modell oder ein bestimmter Datensatz nicht über den Hugging Face Hub verfügbar ist.

Die Tokenizers-Bibliothek von Hugging Face

Bei jedem der bisher gezeigten Pipeline-Beispiele wurde im Hintergrund ein Schritt zur Tokenisierung durchgeführt, bei dem der Rohtext in kleinere Teile bzw. Einheiten, sogenannte Tokens, zerlegt worden ist. Wie das im Einzelnen funktioniert, werden wir in Kapitel 2 erfahren. Für den Moment reicht es, zu wissen, dass Tokens Wörter, Teile von Wörtern oder einfach Zeichen wie Satzzeichen sein können. Transformer-Modelle werden auf Basis numerischer Darstellungen bzw. Repräsentationen dieser Tokens trainiert. Daher ist es für das gesamte NLP-Projekt von großer Bedeutung, diesen Schritt erfolgreich zu meistern!

Die 🤗 Tokenizers-Bibliothek (*https://oreil.ly/Z79jF*) bietet viele verschiedene Strategien zur Tokenisierung und kann Text dank ihres Rust-Backends extrem schnell in Tokens überführen.[13] Sie übernimmt auch alle Vor- und Nachverarbeitungsschritte (engl. Pre/Postprocessing), wie die Normalisierung (engl. Normalization) der Eingaben (engl. Inputs) und die Umwandlung der Ausgaben (engl. Outputs) des Modells in das gewünschte Format. Die Tokenizer der 🤗 Tokenizers-Bibliothek können Sie auf die gleiche Weise laden wie die Modellgewichte vortrainierter Modelle mit der 🤗 Transformers-Bibliothek.

Um Modelle trainieren und auch evaluieren zu können, benötigen wir einen Datensatz und Maße, mit denen wir die Güte- bzw. Qualität des Modells beurteilen können.[14] Werfen wir also einen Blick auf die 🤗 Datasets-Bibliothek, die hierfür vorgesehen ist.

Die Datasets-Bibliothek von Hugging Face

Datensätze zu laden, zu verarbeiten und zu speichern, kann recht mühsam sein, vor allem wenn die Datensätze so groß werden, dass sie nicht mehr in den Arbeitsspeicher Ihres Notebooks passen. Darüber hinaus müssen Sie in der Regel verschiedene Skripte implementieren, mit denen Sie die Daten herunterladen und in ein Standardformat überführen können.

Die 🤗 Datasets-Bibliothek (*https://oreil.ly/959YT*) vereinfacht diesen Prozess, indem sie eine Standardschnittstelle für Tausende von Datensätzen bietet, die auf

13 Rust (*https://rust-lang.org*) ist eine hochgradig leistungsstarke Programmiersprache.

14 Anm. d. Übers.: Zum Zeitpunkt der Übersetzung des Buchs gibt es inzwischen zusätzlich noch die 🤗 Evaluate-Bibliothek (*https://github.com/huggingface/evaluate*), wodurch sich das Vorgehen im Rahmen der Evaluierung leicht verändert hat. Statt z.B. `datasets.load_metric("sacrebleu")` kann inzwischen `evaluate.load("sacrebleu")` verwendet werden. Der im Buch beschriebene Ansatz über die 🤗 Datasets-Bibliothek (*https://oreil.ly/959YT*) ist in den aktuellen Versionen dennoch möglich.

dem Hub (*https://oreil.ly/Rdhcu*) zu finden sind. Außerdem bietet sie eine intelligente Zwischenspeicherung (engl. Caching), sodass die Vorverarbeitung nicht jedes Mal, wenn Sie Ihren Code erneut ausführen, durchgeführt werden muss. Ferner hilft sie dabei, die mit einem begrenzten Arbeitsspeicher verbundenen Einschränkungen zu überwinden, indem sie einen speziellen Mechanismus namens *Memory Mapping* nutzt, bei dem der Inhalt einer Datei im virtuellen Speicher gespeichert und es mehreren Prozessen ermöglicht wird, Dateien auf effizientere Weise zu modifizieren. Die Bibliothek ist außerdem mit gängigen Frameworks wie Pandas und NumPy kompatibel, sodass Sie nicht auf Ihre bevorzugten Tools zur Datenverarbeitung verzichten müssen.

Allerdings sind hochwertige Datensätze und leistungsstarke Modelle wertlos, wenn Sie deren Leistung nicht zuverlässig beurteilen können. Leider gibt es bei den klassischen NLP-Maßen bzw. -Metriken viele verschiedene Implementierungen, die leicht variieren und irreführende Ergebnissen erzielen können. Indem es die Skripte für viele Maße bereitstellt, trägt die 🤗 Datasets-Bibliothek dazu bei, dass Experimente besser reproduzierbar und die Ergebnisse vertrauenswürdiger sind.

Mit der 🤗 Transformers-, der 🤗 Tokenizers- und der 🤗 Datasets-Bibliothek haben wir nun alle notwendigen Voraussetzungen, um unsere eigenen Transformer-Modelle trainieren zu können! Wie wir jedoch noch in Kapitel 10 sehen werden, gibt es Situationen, in denen wir die Trainingsschleife noch stärker modifizieren müssen, als dies die `Trainer`-Klasse zulässt. Hier kommt die letzte Bibliothek des Ökosystems ins Spiel: 🤗 Accelerate.

Die Accelerate-Bibliothek von Hugging Face

Wenn Sie jemals Ihr eigenes Trainingsskript in PyTorch schreiben mussten, dann haben Sie sich wahrscheinlich schon einmal den Kopf darüber zerbrochen, wie Sie den Code, der auf Ihrem Laptop läuft, auf das Cluster Ihres Unternehmens bringen. Die 🤗 Accelerate-Bibliothek (*https://oreil.ly/iRfDe*) fügt Ihren normalen Trainingsschleifen eine zusätzliche Abstraktionsebene hinzu, die sich um die gesamte benutzerdefinierte Logik kümmert, die für die Trainingsinfrastruktur erforderlich ist. Dies vereinfacht eine gegebenenfalls erforderliche Änderung der Infrastruktur und beschleunigt (»accelerates«) somit Ihre Arbeitsabläufe.

Soweit zu den wesentlichen Komponenten des Open-Source-Ökosystems von Hugging Face. Doch bevor wir dieses Kapitel abschließen, werfen wir noch einen Blick auf einige der typischen Herausforderungen, die beim Deployment von Transformer-Modellen in der Praxis auftreten.

Die größten Herausforderungen im Zusammenhang mit Transformer-Modellen

In diesem Kapitel haben Sie einen ersten Eindruck davon bekommen, wie vielfältig die NLP-Aufgaben sind, die mit Transformer-Modellen bewältigt werden können.

Wenn man die Schlagzeilen in den Medien liest, klingt es manchmal so, als seien sie zu allem fähig. Doch so nützlich sie auch sein mögen, Transformer sind keineswegs der Weisheit letzter Schluss. Im Folgenden finden Sie einige mit ihnen einhergehende Herausforderungen, die wir im Laufe des Buchs noch genauer beleuchten werden:

Sprache
: Die Forschung im Bereich des NLP wird überwiegend in englischer Sprache betrieben. Zwar gibt es einige Modelle für andere Sprachen, doch insbesondere für Sprachen, die selten sind oder für die nur wenige Ressourcen zur Verfügung stehen, ist es schwieriger, vortrainierte Modelle zu finden. In Kapitel 4 untersuchen wir mehrsprachige Transformer-Modelle und ihre Fähigkeit, einen sprachübergreifenden Zero-Shot-Transfer durchzuführen.

Datenverfügbarkeit
: Obwohl wir Transfer Learning anwenden können und so die Menge an gelabelten Trainingsdaten, die unsere Modelle benötigen, drastisch verringern können, sind dies im Vergleich zu der Menge, die ein Mensch benötigt, um die Aufgabe zu bewältigen, nach wie vor viele. Wie Sie Situationen meistern können, bei denen Sie nur auf wenige oder gar keine gelabelten Daten zurückgreifen können, erfahren Sie in Kapitel 9.

Mit langen Texten bzw. Dokumenten arbeiten
: Der Self-Attention-Mechanismus funktioniert sehr gut bei Texten, die nur aus einem Absatz bestehen. Bei längeren Texten, z. B. ganzen Dokumenten, ist er jedoch sehr rechenintensiv. Ansätze, die diesem Problem entgegenwirken, finden Sie in Kapitel 11.

Intransparenz (engl. Opacity)
: Wie andere Deep-Learning-Modelle auch, sind Transformer weitgehend intransparent. Es ist schwer oder gar unmöglich, herauszufinden, »warum« ein Modell eine bestimmte Vorhersage getroffen hat. Dies ist eine besonders große Herausforderung, wenn Modelle dazu eingesetzt werden, schwerwiegende Entscheidungen zu treffen. In den Kapiteln Kapitel 2 und Kapitel 4 werden wir einige Möglichkeiten erkunden, wie sich Fehler (engl. Errors) bei Vorhersagen von Transformer-Modellen untersuchen lassen.

Bias bzw. Voreingenommenheit
: Transformer-Modelle werden in erster Linie auf der Grundlage von Textdaten vortrainiert, die aus dem Internet stammen. Dadurch werden alle Vorurteile (engl. Bias), die sich in den Daten widerspiegeln, von den Modellen übernommen. Sicherzustellen, dass diese weder rassistisch noch sexistisch oder anderweitig voreingenommen sind, ist eine anspruchsvolle Aufgabe. Auf einige dieser Probleme gehen wir in Kapitel 10 näher ein.

Diese Herausforderungen mögen zwar entmutigend wirken, doch viele davon können überwunden werden. Wir werden diese Themen nicht nur in den genannten Kapiteln, sondern auch in fast allen anderen Kapiteln ansprechen.

Zusammenfassung

Hoffentlich sind Sie jetzt gespannt darauf, wie Sie diese vielseitigen Modelle trainieren und in Ihre eigenen Anwendungen integrieren können! Sie haben in diesem Kapitel gesehen, dass Sie mit nur wenigen Zeilen Code hochmoderne Modelle zur Klassifizierung, zur Erkennung von Eigennamen (Named Entity Recognition), zur Beantwortung von Fragen (Question Answering), zur Übersetzung und zur Zusammenfassung verwenden können. Doch das ist lediglich die »Spitze des Eisbergs«.

In den folgenden Kapiteln lernen Sie, wie Sie Transformer-Modelle für eine Vielzahl von Anwendungsfällen, wie z.B. für den Aufbau eines Textklassifikators oder eines kompakteren Modells für die Produktion oder sogar für das Training eines Sprachmodells von Grund auf, anpassen können. Dabei werden wir einen praktischen Ansatz verfolgen, d.h., für jedes behandelte Konzept gibt es begleitenden Code, den Sie auf Google Colab oder Ihrem eigenen, mit einer GPU ausgestatteten Rechner ausführen können.

Nachdem wir nun mit den grundlegenden Konzepten hinter Transformern vertraut sind, ist es an der Zeit, dass wir unsere erste Anwendung in Angriff nehmen: die Textklassifizierung, das Thema des nächsten Kapitels.