Fakultät für Wirtschaftswissenschaften, Wirtschaftsinformatik und Wirtschaftsrecht

Studiengang Business Analytics

Masterarbeit

Transformer-Modelle und ihre Anwendungen in der natürlichen Sprachverarbeitung

Autor: Gong Chen

Matrikelnummer: 1471717

Betreuung: Prof. Dr. Ulf Lorenz

Prof. Dr. Marcus Schweitzer

Ort, Abgabetermin: Siegen, 13.06.2023

Abstract

Mit der kontinuierlichen Entwicklung der Natural Language Processing (NLP)-Technologie ist das auf dem Transformer basierende vortrainierte Modell zu einer der Kerntechnologien in diesem Bereich geworden. In dieser Arbeit wird zunächst ein kurzer Überblick über die Entwicklung des NLP gegeben, wobei der Schwerpunkt auf dem Aufkommen vortrainierter Modelle liegt und deren Auswirkungen auf NLP-Aufgaben. Anschließend werden die Grundprinzipien des Transformer-Modells detailliert erläutert, einschließlich seiner Struktur, Schlüsseltechniken sowie Vor- und Nachteile. Danach werden einige repräsentative auf dem Transformer basierende vortrainierte Modelle wie BERT, GPT usw. je nach Architektur aufgelistet und ihre Unterschiede und Verbesserungen verglichen. Schließlich werden die in den letzten Jahren aufgetretenen Large Language Models (LLM) diskutiert und die zukünftige Entwicklung im Bereich NLP beleuchtet. Das Ziel dieser Arbeit ist es, Forschern und Praktikern im Bereich NLP aus der Perspektive der Anwender Informationen über das Transformer-Modell und dessen Anwendung in NLP bereitzustellen.

Inhaltsverzeichnis

1 Einleitung

1.1. Entwicklungsgeschichte der natürlichen Sprachverarbeitung

Die natürliche Sprachverarbeitung (NLP) ist ein wichtiger Bereich der künstlichen Intelligenz, der sich darauf konzentriert, Computern das Verständnis, die Generierung und die Interaktion mit menschlicher natürlicher Sprache zu ermöglichen. Seit den 1950er Jahren hat NLP mehrere Entwicklungsphasen durchlaufen, darunter regelbasierte Ansätze, statistische Ansätze, Deep Learning-Methoden und den Aufstieg vortrainierter Modelle.

1.1.1. Statistische Ansätze

In den 1980er und 1990er Jahren erzielten statistische Ansätze im Bereich NLP signifikante Fortschritte. In dieser Phase lag der Fokus der Forschung auf der Verwendung statistischer Modelle wie dem Hidden Markov Model [1], um Vokabular, Grammatik und Semantik zu modellieren. Ein Vorteil statistischer Ansätze besteht darin, dass sie automatisch Muster aus umfangreichen Korpora lernen können, ohne aufwendige manuelle Regeln festlegen zu müssen. Die Leistung dieser Methoden war jedoch durch die verwendeten Feature-Engineering-Techniken begrenzt, die eine manuelle Auswahl und Gestaltung der Features erforderten.

1.1.2. Deep Learning-Methoden

Zu Beginn des 21. Jahrhunderts begannen sich Deep Learning-Methoden in der NLP durchzusetzen, angetrieben durch verbesserte Rechenleistung und die Verfügbarkeit großer annotierter Datenmengen. In dieser Phase konzentrierte sich die Forschung hauptsächlich auf die Verwendung neuronaler Netzwerke wie Convolutional Neural Networks und Recurrent Neural Networks zur automatischen Erlangung von Textrepräsentationen. Die Einführung von Worteinbettungen wie Word2Vec [2] im Jahr 2013 ermöglichte das Training von Wortrepräsentationen auf großen unmarkierten Datenmengen. Worteinbettungen erzielten bemerkenswerte Erfolge in vielen NLP-Aufgaben und legten den Grundstein für die Entwicklung zukünftiger vortrainierter Modelle. Im Vergleich zu statistischen Ansätzen können Deep Learning-Methoden während des Trainings automatisch effektive Merkmalsrepräsentationen erlernen und haben dadurch zu signifikanten Leistungssteigerungen in vielen NLP-Aufgaben geführt.

1.1.3. Aufstieg vortrainierter Modelle

Im Jahr 2018 erreichte die NLP mit der Einführung des Transformer-Modells [3] und vortrainierter Modelle wie BERT [4] einen neuen Höhepunkt. Vortrainierte Modelle werden durch vortrainieren auf großen unmarkierten Textdaten mit reichhaltigem Sprachwissen versorgt und anschließend für spezifische Aufgaben feinabgestimmt, um signifikante Leistungssteigerungen zu erzielen. Diese Methode hat sich als gängige

Praxis in der heutigen NLP etabliert und hat die Leistungsrekorde in verschiedenen Aufgaben gebrochen.

Im Verlauf der Entwicklung der NLP haben Forscher kontinuierlich nach effektiveren Möglichkeiten gesucht, um Computern das Verständnis und die Verarbeitung natürlicher Sprache zu ermöglichen. Mit zunehmender Rechenleistung und Datenressourcen hat die NLP erhebliche Fortschritte gemacht, insbesondere durch den Fortschritt großer vortrainierter Modelle in den letzten Jahren, die zu bisher unerreichten Leistungen in verschiedenen NLP-Aufgaben geführt haben.

1.2. Erscheinen und Auswirkungen des Transformer-Modells

1.2.1. Innovationen des Transformers

Im Jahr 2017 stellten Vaswani et al. in ihrem Paper "Attention is All You Need" [3] das Transformer-Modell vor, das das Forschungsfeld der natürlichen Sprachverarbeitung grundlegend veränderte. Im Vergleich zu früheren Convolutional Neural Networks (CNNs) und Recurrent Neural Networks (RNNs) enthält das Transformer-Modell folgende wichtige Innovationen:

Self-Attention-Mechanismus: Das Transformer-Modell führt den Self-Attention-Mechanismus ein, der es dem Modell ermöglicht, kontextbezogene Informationen für jede Position in der Eingabesequenz zu berücksichtigen und lange Abhängigkeitsbeziehungen zu erfassen. Durch Multi-Head Attention kann das Modell gleichzeitig verschiedene Repräsentationsräume erlernen, um die Modellkapazität zu verbessern.

Positionskodierung: Da das Transformer-Modell selbst keine zeitliche Informationsverarbeitung besitzt, wird über Positionskodierung die Positionsinformation der Eingabesequenz in das Modell integriert. Dadurch kann das Modell die relativen Positionen zwischen Wörtern erfassen.

Hierarchische Struktur und Residual Connections: Das Transformer-Modell verwendet eine hierarchische Encoder-Decoder-Struktur, wobei jede Schicht aus Multi-Head Attention und einem Feedforward-Netzwerk besteht. Zudem werden Residual Connections und Layer Normalization zwischen den Schichten eingeführt, um den Gradientenfluss und die Stabilität des Trainings zu verbessern.

1.2.2. Auswirkungen des Transformers auf das NLP-Feld

Das Erscheinen des Transformer-Modells hatte weitreichende Auswirkungen auf das NLP-Feld, insbesondere in folgenden Bereichen:

Entwicklung von vortrainierten Modellen: Das Transformer-Modell legte den Grundstein für spätere vortrainierte Modelle wie BERT, GPT [5] und andere. Diese Modelle werden durch vortrainieren auf großen unmarkierten Textdaten mit reichhaltigem Sprachwissen versorgt und anschließend für spezifische Aufgaben

feinabgestimmt, um signifikante Leistungssteigerungen zu erzielen. Diese Methode hat sich als dominierender Ansatz im heutigen NLP-Feld etabliert.

End-to-End-Modellierung: Das Transformer-Modell ermöglicht eine End-to-End-Modellierung und eliminiert komplexe Feature-Engineering-Techniken und Pipeline-Designs. Dadurch besitzt das Modell eine starke Generalisierungsfähigkeit und Skalierbarkeit, was seine Anwendung in verschiedenen NLP-Aufgaben und -Bereichen erleichtert.

Multimodales Lernen und interdisziplinäre Anwendungen: Der Erfolg des Transformer-Modells hat auch die Entwicklung des multimodalen Lernens und die Anwendung in interdisziplinären Bereichen vorangetrieben. Beispielsweise hat das Vision Transformer[6] (ViT) das Transformer-Modell auf Computer-Vision-Aufgaben angewendet und beeindruckende Ergebnisse in der Bildklassifizierung erzielt. Darüber hinaus haben Modelle wie CLIP[7] und DALL-E[8] das Transformer-Modell für multimodales Lernen verwendet, um eine gemeinsame Repräsentation und Generierung von Text und Bildern zu ermöglichen. Diese erfolgreichen Anwendungen haben die starke Ausdrucksfähigkeit und Anpassungsfähigkeit des Transformer-Modells weiter unterstrichen.

Das Erscheinen des Transformer-Modells hat eine Revolution in der natürlichen Sprachverarbeitung ausgelöst und den Grundstein für die Entwicklung von vortrainierten Modellen gelegt. Es hat auch Fortschritte in multimodalem Lernen, Modellkomprimierung und anderen Bereichen vorangetrieben. Der Erfolg des Transformer-Modells zeigt sich nicht nur in Leistungsverbesserungen, sondern auch in seiner breiten Anwendbarkeit, die zu beispiellosem Fortschritt in der Forschung und Anwendung im Bereich NLP geführt hat.

1.3. Ziel und Gliederung der Arbeit

1.3.1 Hauptinhalt und Ziele der Arbeit

Diese Arbeit zielt darauf ab, das Transformer-Modell und seine Anwendungen im Bereich der natürlichen Sprachverarbeitung (NLP) vorzustellen. Zunächst werden die Konzepte, Geschichte und Anwendungen der NLP sowie die Herausforderungen und Entwicklungstrends in diesem Bereich eingeführt. Anschließend wird die Struktur und Funktionsweise des Transformer-Modells detailliert erläutert, und gängige Varianten des Transformer-Modells werden klassifiziert und verglichen. Darüber hinaus wird der Begriff und die Entwicklung von vortrainierten Modellen behandelt, einschließlich Datensätze für Vortraining, Vortraningsaufgaben- und -ansätze. Schließlich werden die jüngsten Fortschritte in großen Sprachmodellen diskutiert.

1.3.2 Überblick über die einzelnen Kapitel

Diese Arbeit besteht aus fünf Kapiteln, die wie folgt gegliedert sind:

Kapitel 1: Einleitung. Dieses Kapitel gibt eine Einführung in die Ziele und die

Gliederung der Arbeit sowie in die Konzepte, Geschichte und Anwendungen der natürlichen Sprachverarbeitung.

Kapitel 2: Transformer-Technologie und Prinzipien. In diesem Kapitel werden zunächst die vorherrschenden Technologien in der NLP vor dem Erscheinen des Transformer-Modells vorgestellt. Anschließend werden die Prinzipien und die Struktur des Transformer-Modells detailliert analysiert, und die Gründe dafür, dass die Transformer-Technologie die vorherigen Ansätze ersetzen kann, werden zusammengefasst.

Kapitel 3: Vortrainierte Modelle basierend auf dem Transformer. In diesem Kapitel werden gängige vortrainierte Modelle basierend auf dem Transformer-Modell entsprechend ihrer Architektur klassifiziert und verglichen.

Kapitel 4: Große Sprachmodelle. In diesem Kapitel werden die Konzepte und die Entwicklung großer Sprachmodelle detailliert erläutert, einschließlich Datensätze für Vortraining, vortrainierter Aufgaben und Feinabstimmung-Verfahren.

Kapitel 5: Zusammenfassung und Ausblick. Dieses Kapitel fasst die Arbeit zusammen und gibt einen Ausblick auf zukünftige Entwicklungen im Bereich der natürlichen Sprachverarbeitung.

Kapitel 2

Von RNN zu Transformer: Evolution neuronaler Netzwerkmodelle in der natürlichen Sprachverarbeitung

Dieses Kapitel wird sich ausführlich mit den Kerntechnologien neuronaler Netzwerkmodelle im Bereich der natürlichen Sprachverarbeitung (NLP) befassen. Im ersten Teil werden wir mit dem dominierenden Modell vor dem Transformer beginnen, dem Rekurrenten Neuronalen Netzwerk (Recurrent Neural Networks, RNN), und untersuchen, wie RNN und seine Varianten Sequenzdaten verarbeiten und komplexe NLP-Probleme lösen können.

Dann werden wir uns dem Transformer-Modell zuwenden. Seit seiner Einführung hat das Transformer-Modell beeindruckende Leistungen in verschiedenen NLP-Aufgaben erbracht. Wir werden die Schlüsselelemente des Transformers, wie die Self-Attention-Mechanismen und die Positionskodierung, eingehend analysieren, um den Lesern ein besseres Verständnis seiner Funktionsweise zu ermöglichen.

Im letzten Teil werden wir ausführlich diskutieren, warum das Transformer-Modell die dominante Position im NLP-Feld erreicht hat und die vorherigen Mainstream-Modelle übertroffen hat. Wir werden analysieren, wie das Transformer-Modell im Vergleich zu RNN die Überlegenheit bei der Verarbeitung von Sequenzdaten, insbesondere von langen Sequenzen, zeigt und wie es durch die Nutzung von Hardware effizientes paralleles Rechnen ermöglicht, was zu herausragender Leistung auf großen Datensätzen führt.

2.1 Neuronale Netzwerke vor dem Erscheinen des Transformers

2.1.1 Einfaches RNN

Das Konzept des rekurrenten neuronalen Netzwerks (Recurrent Neural Network, RNN) entstand in den 1980er Jahren. 1986 stellten David Rumelhart, Geoffrey Hinton und Ronald Williams in ihrer Forschungsarbeit "Learning representations by back-propagating errors" [9] einen effektiven Algorithmus zur Schulung von Netzwerken mit Rückkopplungsverbindungen vor, was das früheste Konzept eines RNN darstellt.

Eine systematischere Untersuchung von RNN erfolgte jedoch erst 1990, als Jeffrey Elman in seinem Artikel "Finding Structure in Time" [10] veröffentlichte. Dies trug zur weiteren Entwicklung von RNN bei. In diesem Artikel stellte Elman eine einfache RNN-Struktur vor, die heute als einfaches RNN bezeichnet wird.

Hier ist ein Diagramm der RNN-Struktur:

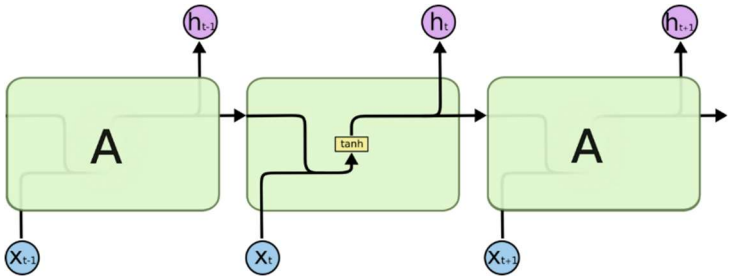


Abb.1 Beispiel eines RNNs [11]

Der Aufbau eines einfachen neuronalen Netzwerks kann in vier Teile unterteilt werden: Eingabe x_i, Ausgabe y_i, versteckter Zustand h_i und die Einheit des rekurrenten neuronalen Netzwerks (RNN-Zelle). Im Vorwärtsberechnungsprozess des rekurrenten Netzwerks ist das RNN eine neuronale Netzwerkeinheit, deren Gewichte und Struktur unverändert bleiben. Diese Netzwerkeinheit erhält bei jeder Berechnung zwei Eingaben: den vorherigen versteckten Zustand h_{i-1} und die aktuelle Eingabe x_i. Sie liefert zwei Ausgaben: den aktuellen versteckten Zustand h_i und den aktuellen Ausgabe y_i.

Für spezifische Deep Learning-Aufgaben besteht die Eingabe x_i oft aus Wortvektoren, die durch eine Worteinbettung-Schicht erzeugt werden.

Der versteckte Zustand des RNNs kann Informationen über die bisher verarbeitete Sequenz erfassen. Für jede Eingabe aktualisiert das RNN seinen versteckten Zustand. Dieser Prozess kann mit folgender mathematischer Formel ausgedrückt werden:

Angenommen, x(t) ist die Eingabe im Zeitschritt t, h(t) ist der versteckte Zustand im Zeitschritt t und o(t) ist die Ausgabe im Zeitschritt t. Dann können wir folgende Gleichungen verwenden:

$h(t) = \tanh(W_{xh} * x(t) + W_{hh} * h(t-1) + b_h)$ [10]

$o(t) = W_{hy} * h(t) + b_y$

Dabei stellt * die Matrixmultiplikation dar, tanh ist die Hyperbelfunktion, W_{xh}, W_{hh} und W_{hy} sind Gewichtsmatrizen, b_h und b_y sind Bias-Vektoren. Die Gewichte und Biases sind die zu lernenden Parameter des Netzwerks während des Trainings.

Die Einführung von RNN hatte einen großen Einfluss auf die natürliche Sprachverarbeitung (NLP) und andere sequenzielle Verarbeitungsaufgaben. Traditionelle neuronale Netzwerke wie Feedforward Neural Networks können keine variablen Eingabesequenzen verarbeiten und berücksichtigen nicht die zeitliche Abfolge der Eingaben. Daher haben sie begrenzte Fähigkeiten bei der Verarbeitung von Sequenzdaten wie natürlicher Sprache.

Im Vergleich dazu ist RNN aufgrund seiner einzigartigen rekurrenten Struktur in der Lage, variablen Eingabesequenzen zu verarbeiten und die zeitliche Abfolge der Eingaben zu berücksichtigen. Daher hat es signifikante Vorteile bei der Verarbeitung von Sequenzdaten wie natürlicher Sprache. Darüber hinaus besitzt RNN die Fähigkeit, vergangene Informationen zu "merken", was seine Leistung bei der Verarbeitung von

zeitlichen oder sequenziellen abhängigen Problemen wie maschineller Übersetzung und Spracherkennung verbessert.

Obwohl RNN diese Vorteile hat, hat es Schwierigkeiten, Abhängigkeiten in langen Sequenzen zu verarbeiten, da es dazu neigt, alte versteckte Zustände bei der Verarbeitung neuer Eingaben allmählich zu vergessen. Dieses Problem wird als "Long-Term Dependency Problem"[12] bezeichnet und wurde in komplexeren RNN-Varianten wie LSTM (Long Short-Term Memory)[12] und GRU (Gated Recurrent Unit)[13] verbessert.

2.2.2 LSTM (Long Short-Term Memory)

Das Long Short-Term Memory (LSTM) ist ein spezielles rekurrentes neuronales Netzwerk (RNN), das erstmals von Sepp Hochreiter und Jürgen Schmidhuber in einem Artikel von 1997 [12] vorgestellt wurde, um das Problem der langfristigen Abhängigkeit in traditionellen RNNs bei der Verarbeitung von langen Sequenzen zu lösen.

Hier ist ein Diagramm der LSTM-Struktur:

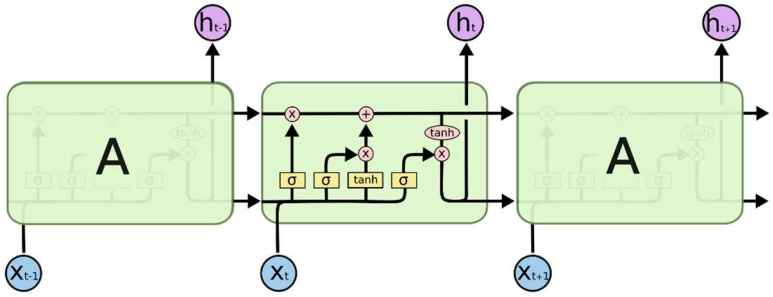


Abb.2 Beispiel eines LSTMs [11]

Der Schlüssel des LSTM liegt in seinem internen Zellzustand (Cell State), der wie ein "Förderband" zwischen den Zeitabschnitten des Netzwerks funktioniert.

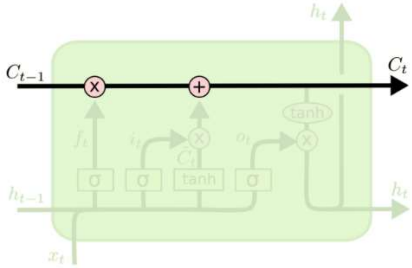
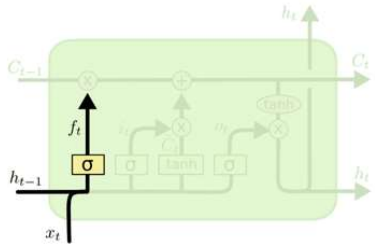


Abb.3 Beispiel eines LSTMs [14]

Informationen können im Zellzustand durch drei verschiedene Arten von Gattern kontrolliert werden: das Vergessensgatter (forget gate), das Eingabegatter (input gate) und das Ausgabegatter (output gate). Jedes Gatter besteht aus einer Sigmoid-Neuronenschicht und einer elementweisen Operationsschicht.

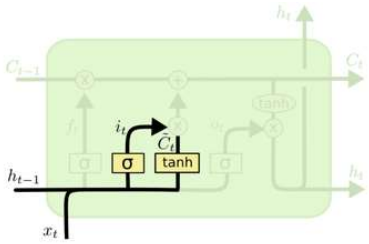
Das Vergessensgatter entscheidet, welche Informationen aus dem Zellzustand verworfen werden sollen. Diese Entscheidung wird durch die Sigmoid-Schicht getroffen, die einen Wert zwischen 0 und 1 ausgibt, der den Prozentsatz der zu behaltenden Informationen darstellt. Die Sigmoid-Schicht betrachtet h(t-1) und x(t) und gibt f(t) aus.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Abb.4 Beispiel eines Vergessensgatters [14]

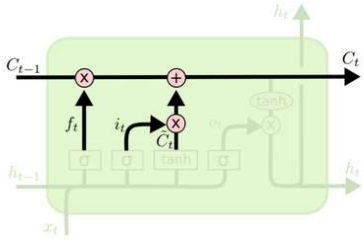
Das Eingabegatter entscheidet, welche Teile des Zellzustands aktualisiert werden sollen. Dieser Prozess besteht aus zwei Teilen: Die Sigmoid-Schicht bestimmt, welche Werte aktualisiert werden sollen, und die tanh-Schicht erzeugt einen neuen Kandidatenwertvektor, der möglicherweise dem Zustand hinzugefügt wird.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Abb.5 Beispiel eines Eingabegatters [14]

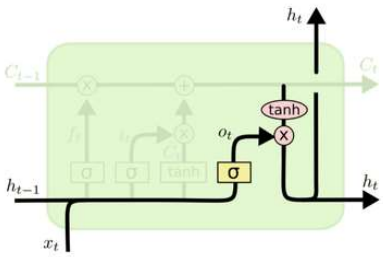
Schließlich aktualisieren wir den alten Zellzustand C_{t-1} zu einem neuen Zellzustand C_t.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Abb.6 Aktualisierung des Zellzustands [14]

Das Ausgabegatter bestimmt, was basierend auf unserem Zellzustand ausgegeben wird. Zuerst entscheidet eine Sigmoid-Schicht, welche Teile des Zellzustands ausgegeben werden sollen. Dann nehmen wir den Zellzustand und wenden die tanh-Funktion darauf an (wobei die Werte zwischen -1 und 1 liegen), multiplizieren ihn mit der Ausgabe des Sigmoid-Gatters und erhalten schließlich die Ausgabe h_t.



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Abb.7 Beispiel eines Ausgabegatters [14]

Hier steht σ für die Sigmoid-Funktion, · für die Matrixmultiplikation und * für das elementweise Produkt. W und b sind Modellparameter.

Diese Gestaltung des LSTM ermöglicht es, längere Abhängigkeiten in Sequenzen zu erfassen und zu speichern. Daher zeigt es eine überlegene Leistung in vielen Aufgaben, die lange Sequenzen erfordern, wie maschinelle Übersetzung, Spracherkennung und andere.

2.1.3 Gate Recurrent Unit (GRU)

Das Gate Recurrent Unit (GRU) wurde von Kyunghyun Cho et al. in ihrem Artikel "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation" [13] im Jahr 2014 vorgestellt. Es handelt sich um eine verbesserte Form des rekurrenten neuronalen Netzwerks (RNN), das entwickelt wurde, um das Problem der langfristigen Abhängigkeiten zu lösen, mit dem traditionelle RNNs konfrontiert sind.

GRU verbessert die Hidden Layer des RNNs und führt zwei neue Gate-Mechanismen ein: das Updategatter und das Resetgatter. Beide Gate-Mechanismen werden mit der Sigmoid-Funktion aktiviert und geben Werte zwischen 0 und 1 aus, um den Informationsfluss zu steuern.

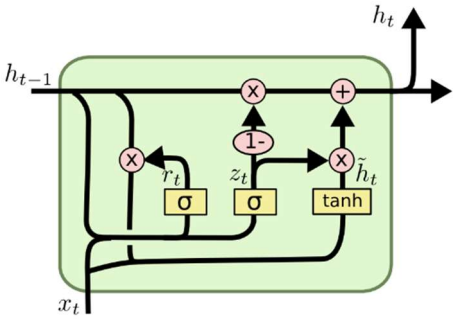


Abb.8 Beispiel eines GRUs [14]

Ähnlich wie bei LSTM hilft das Updategatter dem Modell zu bestimmen, wie viel vergangene Informationen beibehalten werden sollen, wenn der aktuelle Hidden State berechnet wird. Je näher der Wert des Updategatters bei 1 liegt, desto mehr vergangene Informationen werden beibehalten. Das Resetgatter hilft hingegen dem Modell zu bestimmen, wie viel vergangene Informationen beim Berechnen des neuen Hidden

State ignoriert werden sollen. Je näher der Wert des Resetgatters bei 0 liegt, desto mehr vergangene Informationen werden ignoriert. Anschließend wird der Kandidaten-Hidden State $\tilde{h}(t)$ berechnet, der vergangene Hidden States und aktuelle Eingabeinformationen kombiniert. Dieser Schritt ähnelt dem traditionellen RNN, wobei der Hidden State durch den mit dem Resetgatter modulierten Hidden State ersetzt wird. Schließlich wird der aktuelle Hidden State unter Verwendung des Updategatters generiert, indem vergangene Hidden States und der Kandidaten-Hidden State kombiniert werden.

Im Vergleich zu LSTM ist die Modellstruktur von GRU vereinfacht und der Zellzustand, der in LSTM vorhanden ist, wird weggelassen. GRU verwendet nur zwei Gate-Mechanismen anstelle von drei, was zu einer effizienteren Berechnung führt. In einigen Aufgaben, insbesondere wenn die Datenmenge begrenzt ist, kann GRU ähnlich gut oder sogar besser als LSTM abschneiden.

2.2 Transformer-Modellarchitektur

Das Transformer-Modell wurde erstmals von Vaswani et al. im Jahr 2017 in ihrem Artikel "Attention is All You Need" vorgestellt. Im Gegensatz zu traditionellen Sequenz-zu-Sequenz-Modellen, die auf rekurrenten neuronalen Netzwerken (RNN) oder Long Short-Term Memory (LSTM) basieren, verzichtet der Transformer vollständig auf rekurrente Strukturen. Stattdessen verwendet er Mechanismen der Selbst-Aufmerksamkeit (Self-Attention) und Positionsencodierung (Positionskodierung), um die Eingabe- und Ausgabesequenzen zu verarbeiten.

In diesem Abschnitt wird zunächst eine Übersicht über das Modell gegeben, gefolgt von einer schrittweisen Erläuterung der Modellstruktur basierend auf dem Fluss der Daten im Transformer-Modell.

2.2.1 Überblick über die Modellstruktur

Das Transformer-Modell verwendet ein Encoder-Decoder-Struktur. Hier sind zwei schematische Darstellungen der Transformer-Struktur:

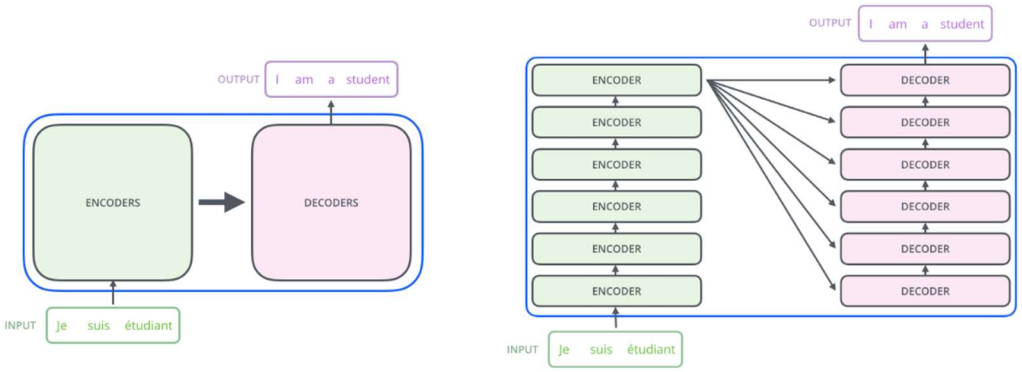


Abb.9 Struktur eines des Transformers [15]

Es ist zu sehen, dass der Transformer aus einem Encoder und einem Decoder besteht,

wobei sowohl der Encoder als auch der Decoder aus 6 Blöcken bestehen. Der Arbeitsablauf des Transformers ist grob wie folgt:

Schritt 1: Erhalten der Repräsentationsvektoren X für jedes Wort im Eingabesatz. X wird durch Addition der Worteinbettung (das aus den Rohdaten extrahierte Merkmal) und des Positionsembeddings für jedes Wort berechnet.

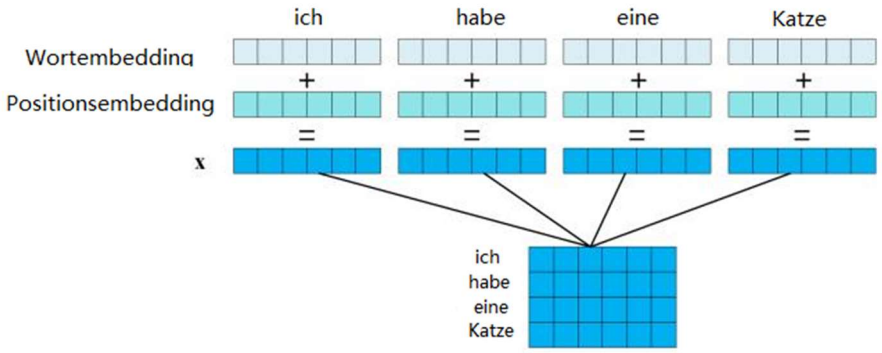


Abb.10 Worteinbettung des Transformers

Schritt 2: Die erhaltenen Wortrepräsentationsvektoren (wie in der obigen Grafik dargestellt, jede Zeile entspricht der Repräsentation x eines Wortes) werden in den Encoder eingegeben. Nachdem sie durch 6 Encoder-Blöcke gegangen sind, erhalten wir die kodierten Informationen für alle Wörter im Satz in Form einer Matrix C, wie im folgenden Diagramm dargestellt.

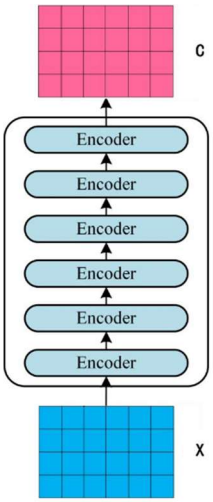


Abb.11 Eingabe und Ausgabe des Encoders

Schritt 3: Die codierte Matrix C, die vom Encoder ausgegeben wurde, wird an den Decoder übergeben. Der Decoder übersetzt nacheinander das nächste Wort i+1 basierend auf den bereits übersetzten Wörtern 1 bis i, wie im folgenden Diagramm dargestellt. Während des Prozesses werden die Wörter ab i+1 mit Hilfe einer Maskenoperation maskiert, um sie zu verbergen.

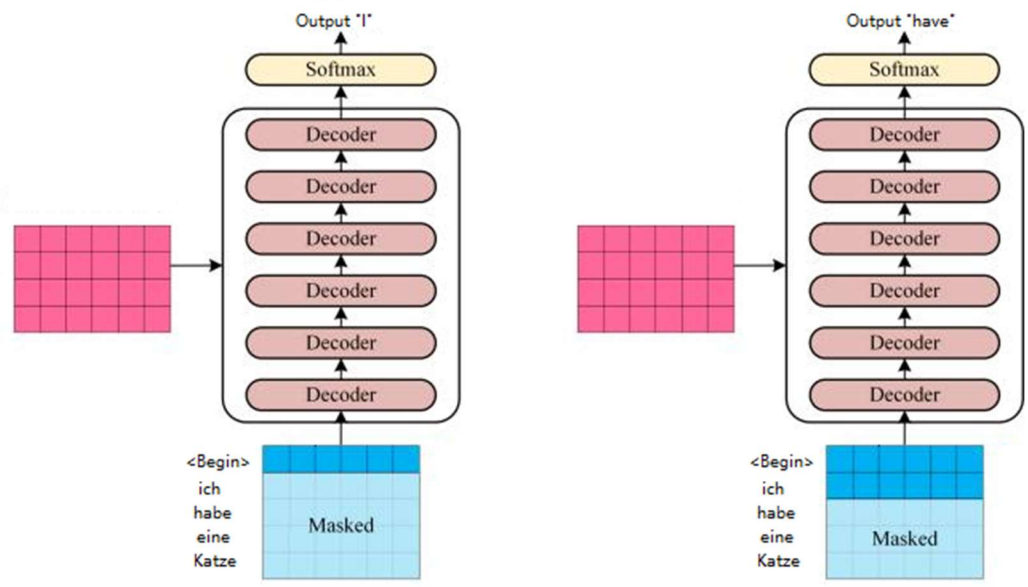


Abb.12 Ausgabe des Decoders

Im obigen Diagramm erhält der Decoder die codierte Matrix C vom Encoder und beginnt zunächst mit einem Übersetzungsstartsymbol "<Begin>". Anschließend wird das erste Wort "I" vorhergesagt. Dann werden das Übersetzungsstartsymbol "<Begin>" und das Wort "I" eingegeben, um das Wort "have" vorherzusagen, und so weiter. Dies ist der Ablauf der Verwendung des Transformers. Im Folgenden werden die Details der verschiedenen Teile erläutert.

2.2.2 Transformer Eingabe (Input)

In der Struktur des Transformers gibt es zwei Teile der Eingabe: Input Embedding und Positional Embedding, die den Modulen 1 und 2 in der Abbildung entsprechen.

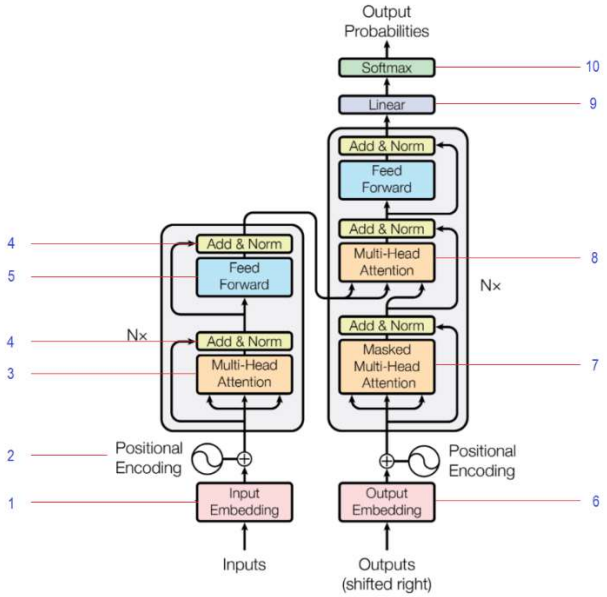


Abb.13 Struktur des Transformers[3]

2.2.3 Transformer Encoder

Input Einbettung (Modul 1)

Im Transformer-Modell wird die Worteinbettung als erster Schritt im Modell durchgeführt. Vor der Ankunft der Textdaten am Modul 1 durchläuft es normalerweise die folgenden beiden Schritte:

1 Tokenisierung: Zunächst müssen die Textdaten in eine Sequenz von Wörtern umgewandelt werden. Dies wird in der Regel durch Tokenisierung erreicht. Für Sprachen wie Englisch, die Leerzeichen als Worttrennzeichen verwenden, können die Sätze direkt anhand von Leerzeichen geteilt werden. Für Sprachen wie Chinesisch, die keine offensichtlichen Worttrennzeichen haben, können komplexere Methoden wie wörterbuchbasierte oder maschinelles Lernen-basierte Ansätze verwendet werden.

2 Kodierung der Wörter: Nach der Tokenisierung erhalten wir eine Sequenz von Wörtern. Dann müssen wir jedes Wort in eine Ganzzahl umwandeln, die in der Regel als Wortindex bezeichnet wird. Dieser Schritt wird in der Regel durch das Nachschlagen in einem Wortschatz (Vocabulary) durchgeführt. Der Wortschatz ist eine Zuordnung aller möglichen Wörter und ihrer entsprechenden Indizes.

Die Schritte Tokenisierung und Kodierung werden **nicht** im Inneren des Transformer-Modells durchgeführt.

Nachdem jedes Wort seinen Index erhalten hat, kann es in einen Vektor umgewandelt werden. Dieser Schritt wird von Modul 1 durchgeführt und erfolgt in der Regel durch das Nachschlagen in einer Wort-Einbettungsmatrix. Jede Zeile der Wort-Einbettungsmatrix stellt eine Einbettung für ein Wort dar, wobei die Anzahl der Zeilen der Größe des Vokabulars entspricht und die Anzahl der Spalten die Dimension der Einbettung angibt.

Beim Initialisieren eines Transformer-Modells wird eine Wort-Einbettungsmatrix benötigt. Diese Matrix wird in der Regel zufällig initialisiert oder mit vorab trainierten Wort-Einbettungen wie Word2Vec, GloVe[16], FastText[17] usw. initialisiert.

Wenn das Training des Transformer-Modells beginnt, wird die Wort-Einbettungsmatrix als Teil der Modellparameter behandelt und zusammen mit den anderen Parametern aktualisiert. Zu diesem Zeitpunkt ändern sich die Werte der Wort-Einbettungsmatrix im Laufe des Trainings.

Positionseinbettungen (Modul 2)

Neben Worteinbettungen erfordert Transformer auch die Verwendung von Positionseinbettungen, um die Positionen der Wörter in einem Satz darzustellen. Da Transformer nicht die sequenzielle Struktur von RNNs verwendet und stattdessen auf globale Informationen angewiesen ist, kann es die Reihenfolge der Wörter nicht direkt erfassen, was in der NLP jedoch entscheidend ist. Daher werden in Transformer Positionseinbettungen verwendet, um die relativen oder absoluten Positionen von

Wörtern in einer Sequenz zu kodieren.

Positionseinbettungen haben die gleiche Dimension wie Wortheinbettungen. Sie können während des Trainings erlernt oder mithilfe einer bestimmten Formel berechnet werden. In Transformer wird letztere Methode verwendet[3], und die Berechnungsformel lautet wie folgt:

$$\overrightarrow{p_t} = \begin{bmatrix} \sin(\omega_1 . t) \\ \cos(\omega_1 . t) \\ \\ \sin(\omega_2 . t) \\ \cos(\omega_2 . t) \\ \\ \vdots \\ \\ \sin(\omega_{d/2} . t) \\ \cos(\omega_{d/2} . t) \end{bmatrix}_{d \times 1}$$

Abb.13 Struktur des Transformers[18]

In der Transformer-Architektur verwendet man Sinus- und Kosinusfunktionen für die Positionskodierung. Der Hauptvorteil dieser Gestaltung besteht darin, dass für jede feste Verschiebung k, Position p und Position p+k eine feste und erlernbare Beziehung im Positionskodierung besteht. Genauer gesagt behalten Sinus- und Kosinusfunktionen eine konsistente Frequenz und Verschiebung für beliebige Positionen p und beliebige Verschiebungen k bei, was es dem Modell ermöglicht, die relativen Positionen in der Sequenz leicht zu erlernen.[18]

Darüber hinaus haben Sinus- und Kosinusfunktionen eine periodische Natur, was bedeutet, dass sie in der Lage sind, lange Sequenzen effektiv zu kodieren, ohne Konflikte in der Kodierung zu verursachen.

Schließlich verändert die Hinzufügung des Positionskodierung aufgrund des begrenzten Ausgabebereichs der Sinus- und Kosinusfunktionen den Bereich der Worteinbettungen nicht wesentlich. Dies ist vorteilhaft für das Lernen und die Optimierung des Modells. Die Verwendung von Sinus- und Kosinusfunktionen als Positionskodierung ermöglicht eine effektive Bereitstellung von Positionsinformationen und gewährleistet gleichzeitig das Lernen und die Optimierung des Modells. Durch Addition der Worteinbettungen und des Positionskodierungs erhält man den Wortrepräsentationsvektor x, der als Eingabe für den Transformer verwendet wird.

Weitere ausführliche Erläuterungen zu dieser Art der Kodierung finden Sie auf folgender Website: https://kazemnejad.com/blog/transformer_architecture_positional_encoding/[18]

Multihead Attention (Modul 3)

Der Aufmerksamkeitsmechanismus ist eine Methode, die in Modellen verwendet wird, um die menschliche visuelle Aufmerksamkeit zu simulieren. Sie ermöglicht es dem Modell, bestimmten Teilen der Eingabe mehr "Aufmerksamkeit" zu schenken. Das Konzept des Aufmerksamkeitsmechanismus wurde erstmals 2014 von Bahdanau et al. in dem Paper "Neural Machine Translation by Jointly Learning to Align and Translate"[19] vorgestellt und wurde hauptsächlich zur Verbesserung der Leistung von maschineller Übersetzung eingesetzt.

Der Aufmerksamkeitsmechanismus wurde entwickelt, um Probleme bei der Verarbeitung von langen Sequenzdaten zu lösen. Traditionelle neuronale Netzwerkmodelle haben oft Schwierigkeiten, Informationen über lange Abstände in einer Sequenz zu erfassen, was als "long-term dependency" bezeichnet wird.

Durch die Verwendung des Aufmerksamkeitsmechanismus kann ein Modell lange Sequenzen effizienter verarbeiten, da es automatisch lernt, welche Teile der Eingabesequenz in jedem Schritt beachtet werden sollten. Dadurch kann das Modell bei Entscheidungen unterschiedliche Gewichtungen für verschiedene Teile der Eingabedaten verwenden und so die Leistung bei der Verarbeitung langer Sequenzen verbessern.[19]

Darüber hinaus bietet der Aufmerksamkeitsmechanismus eine Möglichkeit, die Entscheidungen des Modells zu interpretieren. Die Aufmerksamkeitsgewichte zeigen an, auf welche Teile der Eingabedaten das Modell bei Entscheidungen mehr Aufmerksamkeit richtet, was uns dabei hilft, das Verhalten des Modells zu verstehen und zu erklären.

Self-Attention (Selbstaufmerksamkeit)

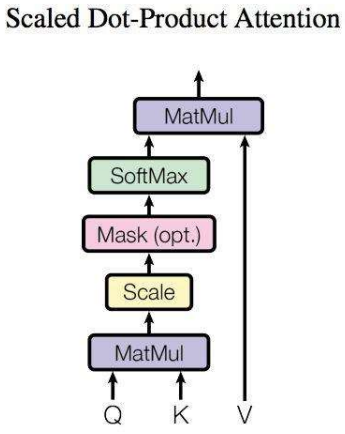


Abb.14 Scaled Dot-Product Attention[3]

Self-Attention (Selbstaufmerksamkeit) ist der Kern des Transformer-Modells und ermöglicht die Kodierung jedes einzelnen Worts in der Eingabesequenz, um eine Reihe von versteckten Zuständen zu erhalten, die die Eingabesequenz repräsentieren. Das obige Diagramm zeigt die Struktur der Self-Attention, bei der Matrizen Q (Abfrage), K (Schlüssel) und V (Wert) verwendet werden. In der Praxis akzeptiert Self-Attention

entweder die Eingabe (eine Matrix X, die aus den Repräsentationsvektoren der Wörter x besteht) oder die Ausgabe des vorherigen Encoder-Blocks. Q, K und V werden durch lineare Transformationen der Self-Attention-Eingabe erhalten.

In einem herkömmlichen Encoder-Decoder-Framework für Aufgaben wie maschinelle Übersetzung sind die Eingabequelle (Source) und die Ausgabeziel (Target) unterschiedlich. Zum Beispiel ist bei der englisch-deutschen maschinellen Übersetzung die Source ein englischer Satz und das Target ist der entsprechende übersetzte deutsche Satz. Die Attention-Mechanismen treten zwischen den Elementen der Query im Target und allen Elementen der Source auf. Im Gegensatz dazu bezieht sich die Selbst-Aufmerksamkeit (Self-Attention) nicht auf die Attention-Mechanismen zwischen Target und Source, sondern auf die Attention-Mechanismen innerhalb der Elemente der Source oder innerhalb der Elemente des Targets. Man kann es auch als Aufmerksamkeitsberechnung innerhalb der Source oder innerhalb des Targets betrachten, wobei der Spezialfall Target = Source einbezogen ist.

In der Multi-Head-Self-Attention (mehrfachen Kopf-Selbstaufmerksamkeit) ist ein entscheidender Schritt die Berechnung der Aufmerksamkeitswerte. Diese Werte bestimmen, wie viel Aufmerksamkeit jedem einzelnen Position gegeben wird, wenn neue Wort-Einbettungsvektoren generiert werden. In Transformer-Modellen werden die Aufmerksamkeitswerte mit der Punktprodukt-Aufmerksamkeit (Dot-Product Attention) berechnet.

Bei der Punktprodukt-Aufmerksamkeit werden gegebene Abfragevektoren Q, Schlüsselvektoren K und Wertvektoren V verwendet. Zuerst wird das Punktprodukt von Q und K berechnet, um die ursprünglichen Aufmerksamkeitswerte zu erhalten. Die Punktprodukt-Operation kann als Maß für die Ähnlichkeit zwischen zwei Vektoren betrachtet werden, da ein größeres Punktprodukt darauf hinweist, dass die Vektoren ähnlicher sind. Anschließend wird der ursprüngliche Aufmerksamkeitswert durch einen Skalierungsfaktor geteilt, der in der Regel die Quadratwurzel der Dimension d der Wort-Einbettungen (sqrt(d)) ist. Dieser Schritt wird als Skalierung bezeichnet. Die Skalierung verhindert, dass die Werte des Punktprodukts zu groß werden und somit die berechneten softmax-Werte zu klein werden, was zu numerischen Stabilitätsproblemen führen kann. Schließlich wird der skalierte Aufmerksamkeitswert durch die softmax-Funktion normalisiert, um die endgültigen Aufmerksamkeitsgewichte zu erhalten.

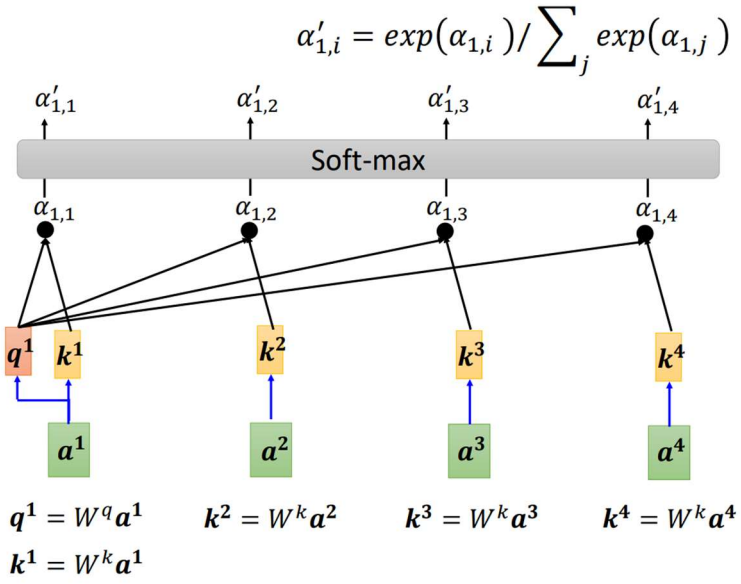


Abb.15 Berechnung der Aufmerksamkeitsgewichte [20]

Nachdem wir die Aufmerksamkeitsgewichte erhalten haben, können wir nun den neuen Wort-Einbettungsvektor für jede Position berechnen. Konkret bedeutet dies, dass wir die Aufmerksamkeitsgewichte mit den entsprechenden Wertvektoren V gewichten und summiert. Auf diese Weise enthält der neue Wort-Einbettungsvektor für jede Position die Informationen der gesamten Sequenz, wobei die Informationen für jede Position durch die Aufmerksamkeitsgewichte gewichtet werden. Das bedeutet, dass das Modell je nach den Anforderungen der Aufgabe selektiv verschiedene Teile der Eingabesequenz beachten kann.

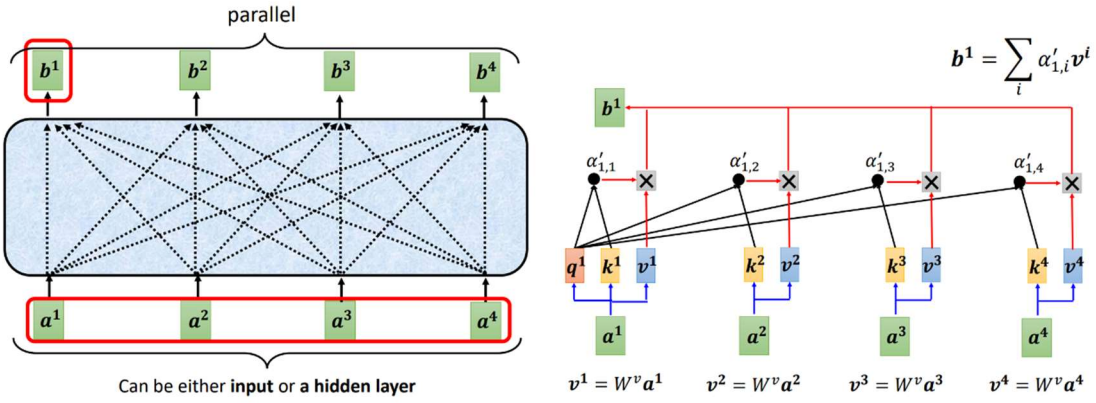


Abb.16 Berechnung des neuen Wort-Einbettungsvektors [20]

Um die Ausdrucksfähigkeit des Modells weiter zu verbessern, verwendet Transformer auch das Multi-Head-Selbst-Aufmerksamkeits-Mechanismus. Dabei wird die Eingangssequenz in verschiedene Projektionsräume projiziert und für jeden Raum eine separate Selbst-Aufmerksamkeitsberechnung durchgeführt. Die Ergebnisse werden schließlich zusammengeführt. Dadurch kann das Modell besser unterschiedliche Merkmale der Eingangssequenz erfassen.

In dem vorherigen Schritt haben wir gelernt, wie man durch Selbst-Aufmerksamkeit die Ausgabematrix berechnet. Multi-Head Attention besteht aus mehreren solcher Selbst-Aufmerksamkeitsmechanismen. Das folgende Diagramm zeigt die Struktur der Multi-Head Attention aus dem Paper.

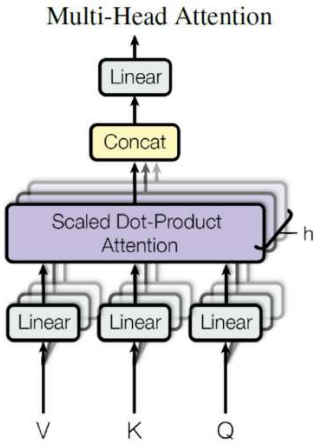


Abb.17 Multihead-Attention [3]

Aus dem obigen Diagramm ist ersichtlich, dass Multi-Head Attention mehrere Selbst-Aufmerksamkeitsschichten enthält. Zunächst wird die Eingabe X über verschiedene lineare Transformationen in h verschiedene Selbst-Aufmerksamkeiten geleitet, um h Ausgabematrizen Z zu berechnen. Das folgende Diagramm zeigt den Fall h = 8, bei dem 8 Ausgabematrizen Z generiert werden.

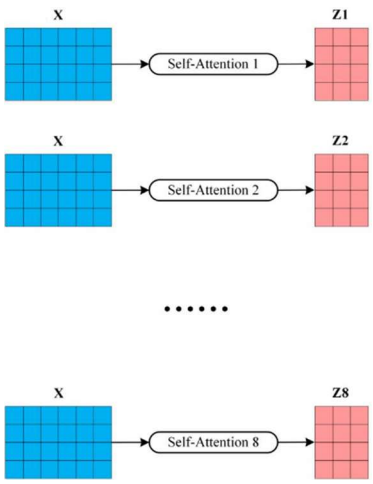


Abb.18 Zwischenschritte der Berechnung der Multihead-Attention

Nachdem die 8 Ausgabematrizen Z1 bis Z8 erhalten wurden, werden sie von Multi-Head Attention zusammengefügt (concatenated) und anschließend durch eine Linearschicht geleitet, um die endgültige Ausgabe Z des Multi-Head Attention zu erhalten.

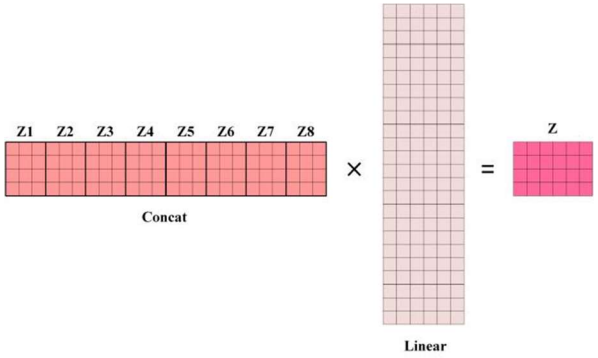


Abb.19 Zusammenfügung der Multihead-Attention Matrizen

Die Ausgabematrix Z des Multi-Head Attention hat die gleiche Dimension wie die Eingabematrix X. Der Vorteil der Multi-Head Attention besteht darin, dass sie dem Modell ermöglicht, verschiedene Merkmale der Eingabesequenz besser zu erfassen, was die Robustheit und Ausdruckskraft des Modells verbessert. Darüber hinaus kann die Berechnung der Multi-Head Attention parallelisiert werden, was die Effizienz des Trainings und der Vorhersage des Modells steigert.

Residualverbindungen[21] und Layer Normalization (Modul 4)

Im Transformer-Modell sind Residualverbindungen (Residual Connections) und Layer Normalization wichtige Designelemente, die die Modellleistung erhalten und gleichzeitig die Stabilität und Effizienz des Trainings verbessern.

Die Schicht "Add & Norm" besteht aus zwei Teilen: Addieren (Add) und Normalisieren (Norm). Die Berechnungsformel lautet wie folgt:

$$\textbf{LayerNorm(X + MultiHeadAttention(X))}$$

$$\textbf{LayerNorm(X + FFN(X))}$$

FFN (Modul 5)

Im Transformer-Modell ist das Feed-Forward-Netzwerk (FFN) ein weiterer wichtiger Bestandteil, das in jedem Transformer-Encoder- und Decoder-Layer enthalten ist.

Das FFN im Transformer-Modell besteht aus zwei linearen Schichten und einer ReLU-Aktivierungsfunktion. Konkret wird bei einer Eingabe von n × d_model (n ist die Sequenzlänge und d_model ist die Dimension der Wort-Einbettungsvektoren) zunächst eine lineare Schicht verwendet, um die Eingabe in eine Zwischenrepräsentation von n × d_ff (d_ff ist die Dimension der versteckten Schicht des FFN, normalerweise größer als d_model) umzuwandeln. Anschließend wird die Zwischenrepräsentation mit der ReLU-Aktivierungsfunktion aktiviert und schließlich durch eine weitere lineare Schicht in eine Ausgabe von n × d_model zurücktransformiert.

Das FFN im Transformer-Modell dient der nichtlinearen Transformation und der Merkmalsextraktion. Es hilft dem Modell dabei, komplexere Muster und Abhängigkeiten in den Eingabesequenzen zu erlernen.[3]

Durch die beschriebenen Module Multi-Head Attention, Feed Forward, Add & Norm kann ein Encoder-Block konstruiert werden. Der Encoder-Block erhält eine Eingabematrix X und gibt eine Ausgabematrix Y zurück. Durch das Stapeln mehrerer Encoder-Blöcke entsteht der gesamte Encoder.

Der erste Encoder-Block erhält eine Eingabematrix der darstellenden Vektoren der Wörter im Satz. Die folgenden Encoder-Blöcke erhalten als Eingabe die Ausgabe des vorherigen Encoder-Blocks. Die Ausgabematrix des letzten Encoder-Blocks ist die Codierungsinformationsmatrix C, die später im Decoder verwendet wird.

2.2.4 Transformer Decoder

Der Aufbau eines Decoder-Blocks ähnelt dem eines Encoder-Blocks, weist jedoch einige Unterschiede auf:

- Es gibt zwei Multi-Head Attention-Schichten.
- Die erste Multi-Head Attention-Schicht verwendet eine maskierte Attention-Operation.
- Die K- und V-Matrizen der zweiten Multi-Head Attention-Schicht werden mit der Codierungsinformationsmatrix C des Encoders berechnet, während Q mit der Ausgabe des vorherigen Decoder-Blocks berechnet wird.
- Schließlich gibt es eine Softmax-Schicht, die die Wahrscheinlichkeit des nächsten übersetzten Worts berechnet.

Die maskierte Multihead-Attention(Modul 7)

Die maskierte Mehrkopf-Selbst-Aufmerksamkeit kombiniert die maskierte Selbst-Aufmerksamkeit mit der Mehrkopf-Aufmerksamkeit. Die Hauptidee besteht darin, die eingehenden Worteinbettungen in mehrere "Köpfe" aufzuteilen und jede Kopf separat zu berechnen. Anschließend werden die Ergebnisse zusammengeführt.

In diesem Prozess, da wir uns im Decoder befinden, wird eine Maskierung eingeführt, um sicherzustellen, dass beim Generieren des aktuellen Wortes keine zukünftigen Informationen berücksichtigt werden. Dies wird durch das Hinzufügen einer Maske während der Berechnung der Selbst-Aufmerksamkeit erreicht, bei der die Elemente der Aufmerksamkeitsgewichtungsmatrix, die die zukünftigen Positionen darstellen, auf negativ unendlich gesetzt werden. Dadurch werden die Auswirkungen zukünftiger Positionen vollständig blockiert, wenn die Softmax-Operation durchgeführt wird.

In der konkreten Implementierung erhält die maskierte Mehrkopf-Selbst-Aufmerksamkeit die eingebettete Repräsentation der Zielsequenz als Eingabe, mit einer Dimension von m×d_model (m ist die Länge der Zielsequenz und d_model ist die Dimension der Worteinbettung). Durch Aufteilung und erneute Kombination erhalten wir h "Köpfe", wobei jeder Kopf eine Dimension von m×(d_model/h) hat. Jeder Kopf führt unabhängig voneinander die Selbst-Aufmerksamkeitsberechnung durch, und die Ergebnisse werden dann zu einer Ausgabematrix der Dimension m×d_model

zusammengeführt.

Die gekreuzte Multihead-Attention(Modul 8)

Die gekreuzte Mehrkopf-Selbst-Aufmerksamkeit ist ein weiterer wichtiger Bestandteil des Decoders. Ihre Funktion besteht darin, dem Decoder ermöglichen, auf die Ausgabe des Encoders zu verweisen und somit die generierten Wörter besser auf die Informationen der Quellsequenz abzustimmen.

In der konkreten Implementierung besteht die Eingabe der gekreuzten Mehrkopf-Selbst-Aufmerksamkeit aus zwei Teilen: der eingebetteten Darstellung der Zielsequenz und der Ausgabe des Encoders. Die eingebettete Darstellung der Zielsequenz wird als "Query" verwendet, während die Ausgabe des Encoders als "Keys" und "Values" verwendet wird. Dies ermöglicht es dem Modell, bei der Generierung jedes Wortes die gesamte Quellsequenz zu berücksichtigen.

2.2.5 Vorhersage des Ausgabeworts

Linear+Softmax (Modul 9 und Modul 10)

Nach der Durchführung einer Reihe von Verarbeitungsschritten durch den Encoder und Decoder erhalten wir schließlich die Ausgabe des Decoders, die verwendet wird, um die endgültige Wortsequenz zu generieren.

In einem Transformer-Modell ist die Ausgabe des Decoders eine Matrix mit der Form m×d_model (m ist die Länge der Zielsequenz und d_model ist die Dimension der Worteinbettung). Diese Matrix enthält eine Darstellung jeder Position der Zielsequenz und enthält bereits Informationen zum Kontext der Quellsequenz.

Um aus dieser Matrix die endgültige Wortsequenz zu generieren, führen wir eine lineare Transformation und eine Softmax-Operation durch. Die lineare Transformation dient dazu, die hochrangige Darstellung jeder Position in einen Vektor der Größe des Vokabulars umzuwandeln, wobei jedes Element des Vektors den Score des entsprechenden Wortes darstellt. Die Gewichtsmatrix für diese lineare Transformation ist die Transposition der Worteinbettung-Matrix.

Nach der linearen Transformation erhalten wir eine Matrix der Form m×V (V ist die Größe des Vokabulars). Anschließend führen wir an jeder Position eine Softmax-Operation durch, um die Scores in Wahrscheinlichkeiten umzuwandeln. Dadurch erhalten wir die Wortverteilung für jede Position, d.h. die Wahrscheinlichkeit, dass jedes Wort an dieser Position erscheint.

2.3 Transformer-Modell: Vor- und Nachteile

2.3.1 Vorteile

Effiziente parallele Berechnung

Das Transformer-Modell mit seiner Self-Attention-Mechanik ermöglicht effiziente

parallele Berechnungen und kann die Rechenleistung moderner Hardware besser nutzen, was die Trainingseffizienz des Modells erhöht. In der Praxis ist das Training eines Transformer-Modells oft wesentlich schneller als bei traditionellen rekurrenten neuronalen Netzen.

Fähigkeit zur Modellierung von langen Abhängigkeiten

Durch die Verwendung von Self-Attention beim Encoding kann das Transformer-Modell Abhängigkeiten zwischen beliebigen Wörtern in der Eingabesequenz erfassen, unabhängig von ihrer Position. Dadurch ist es besser in der Lage, lange Sequenzen zu verarbeiten. Da die Self-Attention alle Interaktionen zwischen den Wörtern berechnet, besitzt das Modell eine bessere Ausdrucksstärke und kann die semantischen Informationen der Eingabesequenz besser erfassen.

Interpretierbarkeit

Die Self-Attention-Mechanik und die Positionscodierung im Transformer-Modell bieten eine hohe Interpretierbarkeit, was dazu beiträgt, die Entscheidungsprozesse und Ausgabewerte des Modells besser zu verstehen. Dadurch können das Modell und seine Ergebnisse besser analysiert und optimiert werden.

2.3.2 Nachteile

Ressourcenanforderungen

Aufgrund seiner großen Modellgröße und der Anzahl der Parameter erfordert das Transformer-Modell erhebliche Rechenressourcen für Training und Inferenz. Dies erschwert den Einsatz des Modells in ressourcenbeschränkten Anwendungsbereichen und erfordert Optimierungen und Komprimierungstechniken.

Große Anzahl von Parametern

Das Transformer-Modell besitzt eine große Anzahl von Parametern, was zu höheren Kosten für Speicherung und Übertragung führt und anfällig für Overfitting sein kann. Daher ist es wichtig, geeignete Regularisierungstechniken und Parameterpruning während des Trainings anzuwenden, um die Generalisierungsfähigkeit und Effizienz des Modells zu verbessern.

Verarbeitung langer Texte

Obwohl das Transformer-Modell Vorteile bei der Verarbeitung langer Sequenzen bietet, können dennoch Herausforderungen bei der Verarbeitung sehr langer Texte auftreten. Beispielsweise können längere Sequenzlängen zu erhöhter Berechnungszeit und Speicherbedarf führen und möglicherweise die Modellleistung und Overfitting-Probleme beeinflussen. Daher müssen angemessene Ansätze zur Segmentierung und Verarbeitung sehr langer Texte angewendet werden, um eine gute Balance zwischen Leistung und Effektivität zu erreichen.

Kapitel 3 Vortrainierte Modelle (Pretrained Language Models)

In den letzten Jahren hat die Verwendung von vortrainierten Sprachmodellen (Pretrained Language Models, PLMs) auf der Basis der Transformer-Architektur eine Revolution im Bereich der natürlichen Sprachverarbeitung (NLP) ausgelöst.

Die Kernidee der vortrainierten Modelle besteht darin, große Mengen an unbeschrifteten Daten für das selbstüberwachte Lernen zu nutzen, um allgemeine sprachliche Eigenschaften wie Grammatik, Syntax und Bedeutung zu erfassen und zu erlernen. Anschließend werden diese Modelle durch einen Feinabstimmungsprozess für spezifische Aufgaben optimiert, um den spezifischen Anforderungen dieser Aufgaben gerecht zu werden. Diese Methode bedeutet im Wesentlichen, dass zuerst ein allgemeines sprachliches Wissen erlernt wird und anschließend eine gezielte Feinabstimmung für spezifische Aufgaben erfolgt.

Diese zweistufige "Vortraining+Feinabstimmung"-Methode hat die Arbeitsweise im Bereich der NLP grundlegend verändert. Traditionelle NLP-Ansätze erforderten in der Regel das manuelle Design und die Extraktion von Merkmalen für jede spezifische Aufgabe, während vortrainierte Modelle automatisch solche Merkmale aus großen Textdaten erlernen können. Diese Veränderung hat nicht nur die Komplexität von Forschung und Entwicklung reduziert, sondern auch die Leistungsfähigkeit verschiedener NLP-Aufgaben erheblich verbessert und somit den Fortschritt in der NLP-Branche vorangetrieben.

3.1 Die Kernbegriffe

3.1.1 Selbstüberwachtes Lernen

Was ist selbstüberwachtes Lernen und wie unterscheidet es sich von unüberwachtem Lernen und überwachtem Lernen?

Selbstüberwachtes Lernen ist ein wichtiger Zweig des Deep Learning, bei dem die Strukturinformationen der Daten selbst als Überwachungssignal verwendet werden. Dabei wird ein Mechanismus entworfen, der es dem Modell ermöglicht, während des Trainings die Eingangsdaten selbst als Labels zu generieren, ohne dass umfangreiche annotierte Informationen von Menschen bereitgestellt werden müssen. Im Trainingsprozess des selbstüberwachten Lernens versucht das Modell, bestimmte Informationen in den Daten vorherzusagen. Zum Beispiel könnte das Modell in der natürlichen Sprachverarbeitung das nächste Wort in einem Satz vorhersagen (wie bei GPT) oder feststellen, ob bestimmte Teile eines Satzes ersetzt wurden (wie bei BERT).

Der Hauptunterschied zwischen überwachtem Lernen und selbstüberwachtem Lernen liegt in der Quelle der Labels. Beim überwachten Lernen müssen für jedes Beispiel menschliche Labels bereitgestellt werden, während beim selbstüberwachten Lernen die Labels aus den Eingangsdaten selbst generiert werden. Daher kann selbstüberwachtes

Lernen als eine Form des überwachten Lernens betrachtet werden, bei dem die Labels automatisch generiert werden.

Der Hauptunterschied zwischen selbstüberwachtem Lernen und unüberwachtem Lernen liegt im Lernziel. Das Ziel des unüberwachten Lernens besteht darin, die inhärente Struktur oder Verteilung der Daten zu entdecken, während beim selbstüberwachten Lernen das Modell durch Vorhersage eines Teils der Daten lernt. Dabei steht die Frage im Vordergrund, wie nützliche Informationen aus einer großen Menge unbeschrifteter Daten gewonnen werden können.

Warum ist selbstüberwachtes Lernen wichtig?

Für Deep Learning-Modelle hängt effektives Lernen in der Regel von einer großen Menge an annotierten Daten ab, deren Beschaffung oft viel Zeit und Ressourcen erfordert. Allerdings gibt es eine große Menge an unbeschrifteten Daten in unserer Welt. Selbstüberwachtes Lernen nutzt diese unbeschrifteten Daten, um nützliche Merkmale und Kenntnisse zu erlernen und verringert damit den Druck der manuellen Annotation von Daten. Gleichzeitig erweitert es den Bereich und die Tiefe des Lernens.

Vortrainierte Sprachmodelle sind ein gutes Beispiel für die Anwendung von selbstüberwachtem Lernen. Diese Modelle werden auf einer großen Menge unbeschrifteter Texte vortrainiert und erlernen Kenntnisse über Grammatik, Syntax, Bedeutung usw. des Textes. Anschließend erfolgt eine überwachte Feinabstimmung für spezifische Aufgaben wie Sentimentanalyse oder benannte Entitätsidentifizierung.

3.1.2 Datensätze für Vortraining

Das Training von vorab trainierten Sprachmodellen basiert in der Regel auf umfangreichen Datensätzen, und die Auswahl des Datensatzes beeinflusst direkt die Sprachfähigkeiten des vorab trainierten Modells.

Da das Hauptziel vorab trainierter Modelle darin besteht, allgemeine Sprachmuster und -merkmale wie Grammatik, Syntax und Wortbedeutung aus einer großen Menge an Textdaten zu lernen, kann ein breit gefächertes, großes und qualitativ hochwertiges Datenset dazu beitragen, dass das vorab trainierte Modell ein reichhaltiges und vielfältiges Sprachwissen sowie präzisere Sprachregeln erlernt.

Die Datensätze können grob in allgemeine Textdaten und spezifische Textdaten unterteilt werden.

Allgemeine Textdaten:

Webseiten: Zum Beispiel der Datensatz Common Crawl[22], der ein öffentlich zugänglicher Datensatz von Webcrawl-Daten ist und Milliarden von Webseiten umfasst. Diese Art von Datensätzen zeichnet sich durch eine breite Abdeckung, verschiedene Sprachstile und Themen aus und kann dazu beitragen, dass das Modell vielfältige Sprachausdrücke und Wissen erlernt.

Bücher: Zum Beispiel der Datensatz BooksCorpus[23], der etwa 8000

unveröffentlichte Bücher enthält. Bücher bieten formellere Langtexte, die dazu beitragen können, dass das Sprachmodell Sprachkenntnisse erlernt und kohärente Texte generiert.

Unterhaltungen: Zum Beispiel der Datensatz Reddit Conversations[24], der eine große Menge an Online-Konversationsaufzeichnungen enthält. Diese Art von Datensätzen zeichnet sich durch einen informelleren Sprachstil aus und enthält reichhaltige soziale und emotionale Informationen.

Spezifische Textdaten:

Mehrsprachige Texte: Zum Beispiel die mehrsprachigen Versionen von Wikipedia[25], die Textdaten in vielen Sprachen enthalten. Diese Art von Datensätzen kann dem Modell bei der Bewältigung von Aufgaben in mehreren Sprachen sehr hilfreich sein.

Domänenspezifische Texte: Zum Beispiel biomedizinische Literatur in PubMed[26] oder wissenschaftliche Artikel in arXiv[27]. Solche Datensätze eignen sich gut zum Training von Modellen mit spezifischem Fachwissen.

Code: Zum Beispiel der GitHub Code-Datensatz[28], der eine große Menge an Programmcode in verschiedenen Programmiersprachen enthält. Dies kann die Fähigkeit des Modells zur automatischen Codegenerierung verbessern.

3.1.3 Vortrainingsaufgaben

Die Vortrainingsaufgaben spielen eine entscheidende Rolle für das Sprachmodell, da sie dem Modell ermöglichen, Sprachkenntnisse aus umfangreichen, unbeschrifteten Daten zu erlernen. Die Leistung des Modells hängt einerseits von der Qualität der Datensätze für Vortraining ab und andererseits davon, ob das Modell durch effektives Vortraining aus den Korpora Sprachregeln und -merkmale gelernt hat. Hier sind einige gängige Vortrainingsaufgaben:

Language Model (LM)

Das Language-Model-Aufgabe ist eine der frühesten Vortrainingsaufgaben, bei der das Modell auf der Grundlage einer gegebenen Wortsequenz das nächste Wort vorhersagen soll. Während des Vortrainings wird das Modell auf einer großen Textmenge trainiert, um die statistischen Sprachregeln zu erlernen.

Masked Language Model (MLM)

Bei der MLM-Aufgabe wird ein Teil der Eingabesequenz zufällig durch ein spezielles MASK-Tag ersetzt, und das Modell muss das ursprüngliche Wort vorhersagen, das ersetzt wurde. Diese Aufgabenstellung wurde erstmals im BERT-Modell eingeführt und hat sich als effektiv erwiesen, um die Leistung des Modells zu verbessern. Der Vorteil von MLM besteht darin, dass kontextuelle Informationen vollständig genutzt werden können, jedoch besteht der Nachteil darin, dass es in echten Downstream-Aufgaben aufgrund des fehlenden Zugriffs auf MASK-Tags zu Diskrepanzen zwischen Vortraining und Feinabstimmung kommen kann.

Permutation Language Model (PLM)

Die PLM-Aufgabe basiert auf MLM und beinhaltet nicht nur die MASK-Behandlung der Eingabesequenz, sondern auch eine zufällige Permutation der Sequenz. Diese Aufgabenstellung wurde erstmals im XLNet-Modell[29] vorgestellt.

Denoising Autoencoder (DAE)

Das DAE-Ziel besteht darin, die ursprüngliche unverzerrte Sequenz basierend auf einer Eingabesequenz mit eingefügtem Rauschen zu rekonstruieren. Diese Aufgabenstellung wird in Modellen wie T5[30] weit verbreitet angewendet, um verschiedene Arten von Rauschen wie Deletion, Ersetzung und Umordnung zu behandeln.

Deep InfoMax (DIM)

Das DIM-Ziel besteht darin, die gegenseitige Information zwischen globaler Information und lokaler Information zu maximieren. Diese Aufgabenstellung hilft dem Modell, tiefere semantische Merkmale zu erlernen.

Next Sentence Prediction (NSP) / Sentence Order Prediction (SOP)

Bei NSP und SOP wird das Ziel verfolgt, die Beziehung zwischen zwei Sätzen vorherzusagen. Bei NSP muss das Modell vorhersagen, ob der zweite Satz der nächste Satz des ersten Satzes ist. Bei SOP muss das Modell die Reihenfolge von zwei Sätzen vorhersagen. Diese Aufgabenstellung hilft dem Modell, die Beziehung und Logik zwischen Sätzen zu erlernen.

Replaced Token Detection (RTD)

Bei der RTD-Aufgabe soll das Modell die ersetzten Tokens in der Eingabesequenz erkennen. Diese Aufgabenstellung wurde erstmals im ELECTRA-Modell[31] eingeführt, bei dem die ursprüngliche Vorhersageaufgabe in eine Klassifikationsaufgabe umgewandelt wird, um das Training des Modells zu beschleunigen.

Die folgende Tabelle fasst gängige Vortrainingsaufgaben und ihre Verlustfunktionen zusammen:

Task	Loss Function	Description
LM	$\mathcal{L}_{\text{LM}} = -\sum_{t=1}^{T} \log p(x_t \mid \mathbf{x}_{<t})$	$\mathbf{x}_{<t} = x_1, x_2, \cdots, x_{t-1}$.
MLM	$\mathcal{L}_{\text{MLM}} = -\sum_{\hat{\mathbf{x}} \in m(\mathbf{x})} \log p\left(\hat{\mathbf{x}} \mid \mathbf{x}_{\setminus m(\mathbf{x})}\right)$	$m(\mathbf{x})$ and $\mathbf{x}_{\setminus m(\mathbf{x})}$ denote the masked words from $\mathbf{x}$ and the rest words respectively.
Seq2Seq MLM	$\mathcal{L}_{\text{S2SMLM}} = -\sum_{t=i}^{j} \log p\left(x_t \mid \mathbf{x}_{\setminus \mathbf{x}_{i:j}}, \mathbf{x}_{i:t-1}\right)$	$\mathbf{x}_{i:j}$ denotes an masked n-gram span from i to j in $\mathbf{x}$.
PLM	$\mathcal{L}_{\text{PLM}} = -\sum_{t=1}^{T} \log p(z_t \mid \mathbf{z}_{<t})$	$\mathbf{z} = perm(\mathbf{x})$ is a permutation of $\mathbf{x}$ with random order.
DAE	$\mathcal{L}_{\text{DAE}} = -\sum_{t=1}^{T} \log p(x_t \mid \hat{\mathbf{x}}, \mathbf{x}_{<t})$	$\hat{\mathbf{x}}$ is randomly perturbed text from $\mathbf{x}$.
DIM	$\mathcal{L}_{\text{DIM}} = s(\hat{\mathbf{x}}_{i:j}, \mathbf{x}_{i:j}) - \log \sum_{\hat{\mathbf{x}}_{i:j} \in \mathcal{N}} s(\hat{\mathbf{x}}_{i:j}, \hat{\mathbf{x}}_{i:j})$	$\mathbf{x}_{i:j}$ denotes an n-gram span from i to j in $\mathbf{x}$, $\hat{\mathbf{x}}_{i:j}$ denotes a sentence masked at position i to j, and $\bar{\mathbf{x}}_{i:j}$ denotes a randomly-sampled negative n-gram from corpus.
NSP/SOP	$\mathcal{L}_{\text{NSP/SOP}} = -\log p(t \mid \mathbf{x}, \mathbf{y})$	$t = 1$ if $\mathbf{x}$ and $\mathbf{y}$ are continuous segments from corpus.
RTD	$\mathcal{L}_{\text{RTD}} = -\sum_{t=1}^{T} \log p(y_t \mid \hat{\mathbf{x}})$	$y_t = \mathbf{1}(\hat{x}_t = x_t)$, $\hat{\mathbf{x}}$ is corrupted from $\mathbf{x}$.

[1] $\mathbf{x} = [x_1, x_2, \cdots, x_T]$ denotes a sequence.

Tabelle 1 Zusammenfassung der Vortrainingsaufgaben[32]

3.2 PLMs basierend auf der Transformer-Architektur

In diesem Abschnitt werden einige Modelle vorgestellt, die einen wichtigen Einfluss auf die Entwicklung von PLMs hatten. Diese Modelle lassen sich grob in drei Kategorien einteilen: Modelle basierend auf dem Encoder, Modelle basierend auf dem Decoder und Modelle basierend auf dem Encoder-Decoder. Jede Kategorie hat ihre repräsentativen Modelle und Schlüsseltechniken, die jeweils ihre eigenen Vorteile und Merkmale aufweisen. Die Auswahl der Architektur hängt von den spezifischen Anwendungsszenarien und Aufgabenanforderungen ab. Im Folgenden werden wir diese drei Kategorien genauer erläutern.

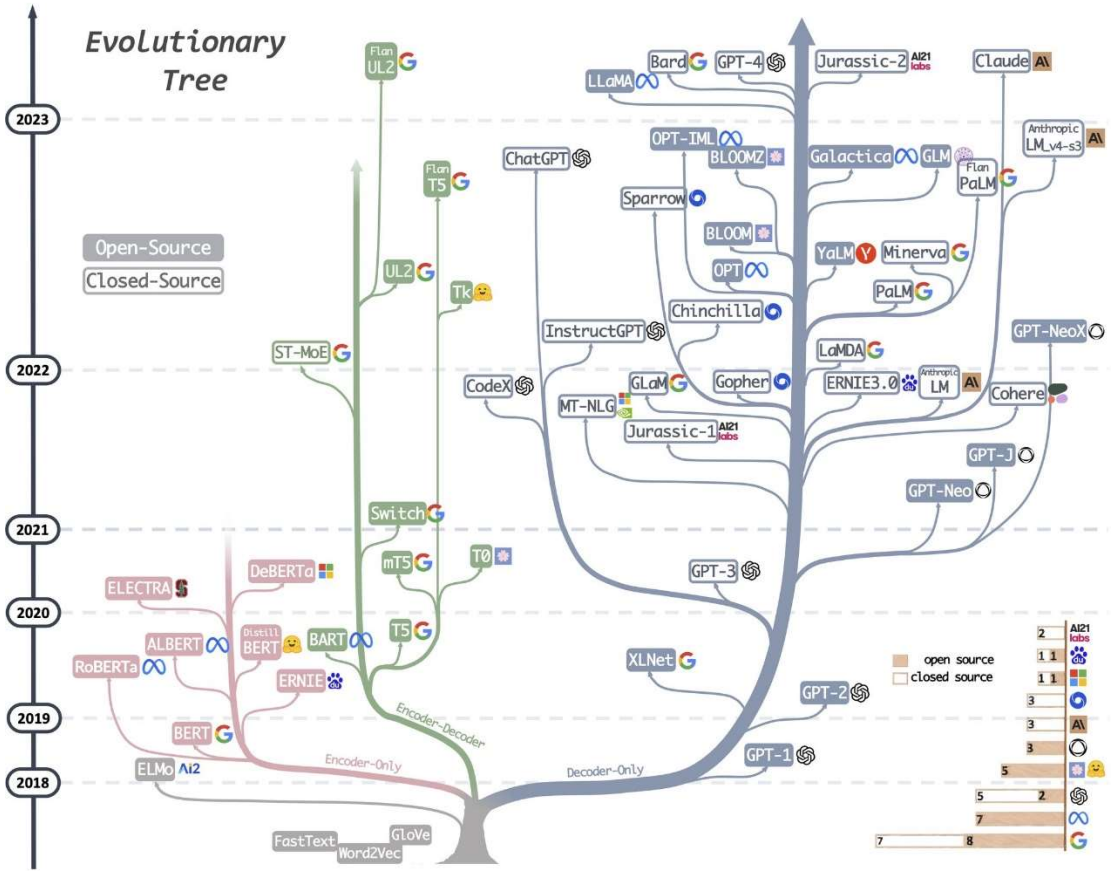


Abb.20 Evolutionary Tree der PLMs[33]

3.2.1 Modelle basierend auf dem Encoder

Modelle basierend auf dem Encoder konzentrieren sich hauptsächlich auf die Extraktion nützlicher Informationen aus den Eingabedaten. Sie verwenden in der Regel die Encoder-Struktur des Transformers. Diese Modelle eignen sich gut für Aufgaben wie Textklassifikation, benannte Entitätenerkennung usw., da diese Aufgaben lediglich erfordern, Merkmale aus den Eingabedaten zu extrahieren, anstatt neuen Text zu generieren.

3.2.1.1 BERT

BERT hat eine große Bedeutung für NLP, da es gezeigt hat, dass ein sehr tiefschichtiges Modell die Genauigkeit von NLP-Aufgaben signifikant verbessern kann und dieses Modell aus unmarkierten Datensätzen vorab trainiert werden kann. Mit BERT hat sich die Forschung und Anwendung von NLP in ein neues Paradigma verwandelt: Vortrainingsmodelle + Feinabstimmung. Vor BERT mussten wir für jede NLP-Aufgabe komplexe Modelle mit schlechter Generalisierungstauglichkeit tiefgreifend anpassen. Aber da BERT bereits ausreichend sprachliche Informationen enthält, reicht es aus, für jede Aufgabe eine leichtgewichtige Ausgabeschicht (z. B. eine einzige MLP-Schicht) anzupassen. Auf diese Weise hat BERT den damaligen Rekord für die 11 Sprachaufgaben im GLUE-Testdatensatz[39] gebrochen.

System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$BERT_{BASE}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$BERT_{LARGE}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Tabelle 2 GLUE-Benchmark von BERT[4]

Die **Architektur** von BERT basiert auf der ursprünglichen Implementierung des mehrschichtigen bidirektionalen Transformer-Encoders, wie in [3] beschrieben. Die Modellstruktur des Transformers wurde in einem früheren Abschnitt ausführlich erläutert.

In dieser Arbeit werden die Anzahl der Schichten (Transformer-Blöcke) als L, die versteckte Größe als H und die Anzahl der Self-Attention-Köpfe als A bezeichnet. In allen Fällen wird die Größe des Feed-Forward-Filters als 4H festgelegt, d.h. 3072 für H = 768 und 4096 für H = 1024. Die Ergebnisse wurden hauptsächlich für zwei Modellgrößen gemeldet:

BERTbase: L=12, H=768, A=12, Gesamtzahl der Parameter = 110 Millionen

BERTlarge: L=24, H=1024, A=16, Gesamtzahl der Parameter = 340 Millionen[4]

Für den Vergleich wurde BERTlarge gewählt, da es die gleiche Modellgröße wie OpenAI GPT hat. Es ist jedoch wichtig zu beachten, dass der BERT-Transformer eine bidirektionale Selbst-Aufmerksamkeit verwendet, während der GPT-Transformer eine eingeschränkte Selbst-Aufmerksamkeit hat, bei der jedes Token nur auf den linksseitigen Kontext achtet. Als Vergleichsstruktur gibt es auch das ELMo-Modell[40], das ebenfalls eine bidirektionale Struktur verwendet, jedoch LSTM zur Merkmalsextraktion der Eingabesequenz einsetzt und weniger Schichten als BERT hat.

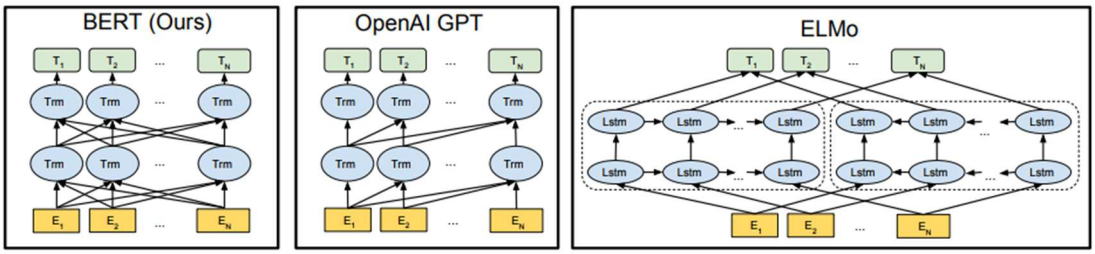


Abb.21 Vergleich Modelle von ELMo,BERT und GPT[4]

Die **Eingaberepräsentation** (Input Representation) von BERT ermöglicht es, einzelne Textsätze oder Satzpaare (z.B. [Frage, Antwort]) in einer Token-Sequenz explizit darzustellen. Für jedes Token wird die Eingaberepräsentation durch die Summe der entsprechenden Token-, Segment- und Positionsembettungen konstruiert.

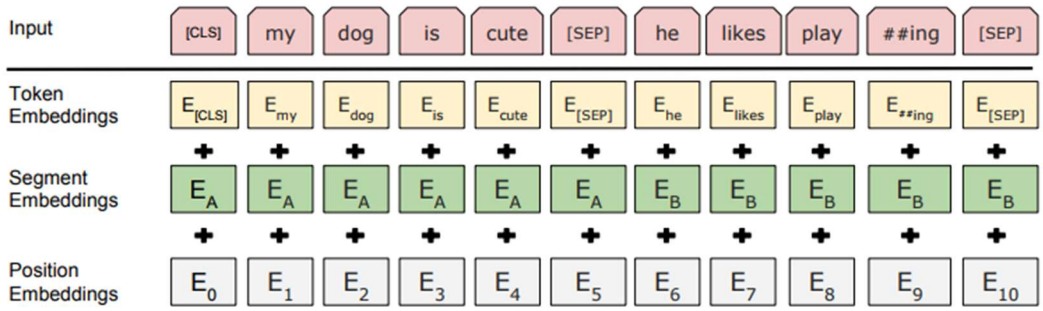


Abb.22 Eingaberepräsentation von BERT[4]

(1) Es wird die Verwendung von WordPiece-Embeddings [34] mit einem Vokabular von 30.000 Tokens verwendet.

(2) Es werden gelernte Positionsembettungen verwendet, wobei die unterstützte maximale Sequenzlänge 512 Tokens beträgt. Das erste Token jeder Sequenz ist immer ein spezielles Klassifikations-Embedding ([CLS]). Der endgültige Hidden State (d.h. die Ausgabe des Transformers), der diesem Token entspricht, wird als aggregierte Sequenzrepräsentation für die Klassifikationsaufgabe verwendet. Für nicht-klassifikationsbasierte Aufgaben wird dieser Vektor ignoriert.

(3) Satzpaare werden zu einer einzigen Sequenz zusammengepackt. Die beiden Sätze werden auf zwei Arten voneinander unterschieden. Erstens werden sie mit einem speziellen Trennungs-Token ([SEP]) getrennt. Zweitens wird jedem Token des ersten Satzes ein Satz-A-Embedding hinzugefügt, und jedem Token des zweiten Satzes wird ein Satz-B-Embedding hinzugefügt.

(4) Für einzelne Satzeingaben wird nur das Satz-A-Embedding verwendet.

Vortrainingsaufgaben:

Aufgabe 1: Maskierte Sprachmodellierung (Masked LM)

Um eine tiefe bidirektionale Repräsentation zu trainieren, verwendet BERT eine einfache Methode, bei der 15 % der Eingabetoken zufällig maskiert werden. Die endgültigen Hidden-Vektoren, die den maskierten Token entsprechen, werden in einem Softmax über das Vokabular zur Vorhersage verwendet. Es gibt jedoch ein Problem mit dieser Methode, nämlich die Unstimmigkeit zwischen Pretraining und Feinabstimmung, da während der Feinabstimmung keine [MASK]-Token vorhanden sind. Um dieses Problem zu lösen, führt der Daten-Generator die folgenden Operationen aus, anstatt das ausgewählte Wort immer durch [MASK] zu ersetzen:

80 % der Zeit: Das Wort wird durch das [MASK]-Token ersetzt.

10 % der Zeit: Das Wort wird durch ein zufälliges Wort ersetzt.

10 % der Zeit: Das Wort bleibt unverändert.

Aufgabe 2: Satzvorhersage (Next Sentence Prediction)

Viele wichtige Aufgaben, wie Frage-Antwort (QA) und natürliche Sprachverarbeitung (NLI), basieren auf dem Verständnis der Beziehung zwischen zwei Sätzen, was nicht direkt durch Sprachmodellierung erfasst wird. Um ein Modell zu trainieren, das die Beziehung zwischen Sätzen versteht, wurde diese Trainingsaufgabe definiert.

Feinabstimmung Phase:

Für Aufgaben der Satzpaarklassifikation wie MNLI-Textimplikation oder QQP, bei denen beurteilt wird, ob zwei Sätze sich wiederholen, werden die Sätze zuerst mit speziellen Token CLS und SEP in das Modell eingegeben und dann anhand der BERT-Repräsentation des Tokens CLS klassifiziert.

Für Einzelsatz-Aufgaben wie SST-2 Stanford Sentiment Treebank oder CoLA, bei denen beurteilt wird, ob ein Satz grammatikalisch korrekt ist, erfolgt die Klassifizierung ebenfalls anhand der BERT-Repräsentation des Tokens CLS.

Für Frage-Antwort-Aufgaben wird für jeden Token im gegebenen Abschnitt eine Vorhersage für den Start- und Endpunkt gemacht, um einen Bereich als Antwort auf die Frage zu finden.

Für die Sequenzmarkierung, wie Named Entity Recognition, wird anhand der entsprechenden BERT-Repräsentation für jeden Token die Vorhersage des Labels gemacht.[4]

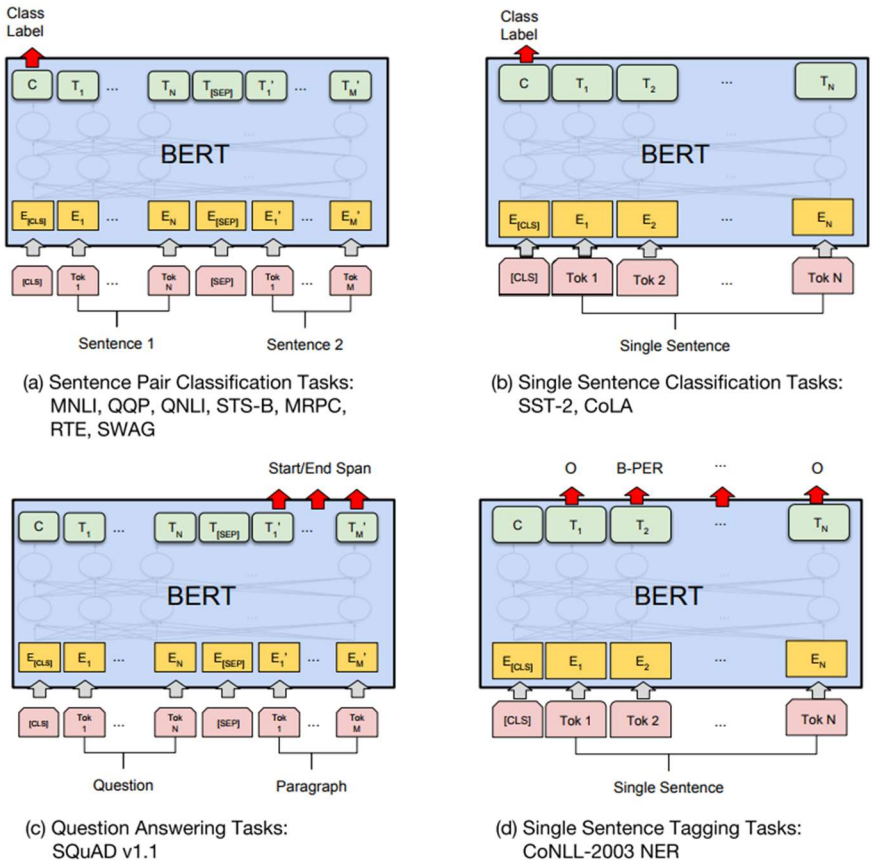


Abb.23 Feinabstimmung von BERT[4]

3.2.1.2 RoBERTa (Ein robust optimierter BERT Pretraining-Ansatz) [41]

Nach der Einführung von BERT gab es einige nachfolgende Modelle (wie XLNet), die die Leistung von BERT weiter verbessert haben. In diesem Paper wird argumentiert, dass BERT nicht ausreichend trainiert wurde und daher seine Leistung besser sein könnte. Aus diesem Grund wurde RoBERTa entwickelt, indem das Modell, die Trainingstrategie und die verwendeten Daten in Betracht gezogen wurden.

Verbesserungen:

1 Mehr Daten

BooksCorpus + English Wikipedia (16 GB): Die Daten, die auch von BERT verwendet wurden.

CC-News[35] (76 GB): Aus den CommonCrawl News-Daten ausgewählte Daten, die etwa 63 Millionen Nachrichtenartikel aus dem Zeitraum von September 2016 bis Februar 2019 umfassen.

OpenWebText[36] (38 GB): Diese Daten wurden aus dem Reddit-Forum übernommen, wobei nur Inhalte mit einer Bewertung von mehr als 3 verwendet wurden, und dienen als Inspiration von GPT-2[37].

Stories (31 GB): Ebenfalls aus CommonCrawl extrahierte Daten, die fiktive Geschichten enthalten.

2 Größere Batchgröße

RoBERTa hat während des Trainings größere Batchgrößen ausprobiert, die von 256 bis 8000 variierten.

3 Verbesserungen der Trainingsmethode

Entfernung der NSP-Trainingsaufgabe

Dynamische Maskierung: Im Gegensatz zur statischen Maskierung des ursprünglichen BERT verwendet RoBERTa eine dynamische Maskierung. Bei jeder Eingabe eines Sequenzes wird ein neues Maskierungsmuster generiert. Dadurch kann das Modell im Laufe der Zeit verschiedene Maskierungsstrategien anpassen und unterschiedliche Sprachrepräsentationen erlernen.

Textkodierung: Die ursprüngliche BERT-Implementierung verwendet ein zeichenbasiertes BPE-Vokabular mit einer Größe von 30.000. Dieses Vokabular wird nach einer heuristischen Tokenisierungsregel auf den Eingabedaten trainiert. Die Facebook-Forscher haben sich jedoch für die Verwendung eines größeren Byte-basierten BPE-Vokabulars entschieden, das 50.000 Subword-Einheiten enthält und keine zusätzliche Vorverarbeitung oder Tokenisierung der Eingabe erfordert.[41]

Model	data	bsz	steps	SQuAD (v1.1/2.0)	MNLI-m	SST-2
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**
$\text{BERT}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7
$\text{XLNet}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	94.0/87.8	88.4	94.4
+ additional data	126GB	2K	500K	94.5/88.8	89.8	95.6

Tabelle 3 : Vergleich Trainingsergebnisse von RoBERTa [41]

In dem Paper wurden die Trainingsergebnisse von RoBERTa, BERT und XLNet verglichen. Alle drei Modelle wurden mit dem Books + Wiki-Datensatz trainiert. Durch den Vergleich wurde festgestellt, dass RoBERTa nach den Verbesserungen im Vergleich zu BERT deutliche Verbesserungen aufweist.

Nach den oben genannten Verbesserungen erzielte RoBERTa die besten Ergebnisse in den Tests von GLUE, SQuAD und RACE.

RoBERTa übertraf sowohl BERT als auch XLNet und zeigte eine signifikante Leistungssteigerung.

3.2.1.3 ALBERT（A LITE BERT）[42]

ALBERT ist eine abgespeckte Version von BERT mit einer deutlich reduzierten Parameteranzahl. Obwohl ALBERT weniger Parameter hat, erreicht es vergleichbare Leistung wie die traditionelle BERT-Architektur.

ALBERT ist ein leichtgewichtiges Modell, das eine effizientere Nutzung der Ressourcen ermöglicht, ohne die Performance einzuschränken.

Model		Parameters	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg	Speedup
BERT	base	108M	90.4/83.2	80.4/77.6	84.5	92.8	68.2	82.3	4.7x
	large	334M	92.2/85.5	85.0/82.2	86.6	93.0	73.9	85.2	1.0
ALBERT	base	12M	89.3/82.3	80.0/77.1	81.6	90.3	64.0	80.1	5.6x
	large	18M	90.6/83.9	82.3/79.4	83.5	91.7	68.5	82.4	1.7x
	xlarge	60M	92.5/86.1	86.1/83.1	86.4	92.4	74.8	85.5	0.6x
	xxlarge	235M	**94.1/88.3**	**88.1/85.1**	**88.0**	**95.2**	**82.3**	**88.7**	0.3x

Tabelle 4 : Vergleich Pamameter von BERT und ALBERT [42]

ALBERT überwindet die Hauptprobleme bei der Skalierung von Vortrainingsmodellen durch zwei Parameterreduktionstechniken.

Parameterisierung der aufgeteilten Einbettungen

In Modellen wie BERT und RoBERTa haben die WordPiece-Einbettungen dieselbe Größe wie die Hidden Size. Die Hidden Size bezieht sich auf die Hidden Size im Transformer-Encoder, also E=H.

Für das Modell lernt die WordPiece-Einbettungsschicht kontextunabhängige Repräsentationen, während die Hidden Layer kontextabhängige Repräsentationen lernt, die deutlich komplexer sind. In der Regel sollte H also größer sein als E. Da das Vokabular V sehr groß ist, würde die Einbettungsmatrix sehr groß werden, wenn E mit zunehmendem H wächst.

Im Paper wird vorgeschlagen, die Encoder-Schicht zu faktorisieren, um die Beziehung zwischen E und H zu durchbrechen. Konkret wird die Einbettungsmatrix in zwei Matrizen VE und EH aufgeteilt, sodass die Berechnung der Einbettungsmatrix bei großen H-Werten effizienter wird.

Die Autoren haben verschiedene Werte für E ausprobiert (64, 128, 256, 768) und eine unabhängige, kontextunabhängige Einbettung (V*E) trainiert, die dann bei der Berechnung in den Hidden-Layer-Raum projiziert wird (multipliziert mit einer E*H-Matrix). Dadurch reduziert sich die Gesamtzahl der Einbettungsparameter von V*H auf V*E+E*H. Wenn E deutlich kleiner als H ist, werden die benötigten Modellparameter erheblich reduziert. Experimente haben gezeigt, dass E=128 die beste Leistung erzielt.

Cross-layer parameter sharing

Die Parameterisierung der aufgeteilten Einbettungen ist nicht der Hauptbeitrag zur

Reduzierung der Modellparameter in ALBERT. Stattdessen ist das Cross-layer parameter sharing (Parameter-Sharing über die Schichten) dafür verantwortlich, den Großteil der Parameter in BERT zu reduzieren. Der Mechanismus des Cross-layer parameter sharing ist sehr einfach: Es wird eine einzelne Self-Attention-Schicht verwendet, die 12-mal wiederholt wird, wobei die Parameter jeder Schicht identisch sind. Dadurch werden die Parameter von 12 Schichten mit den Parametern einer Schicht repräsentiert, was zu einer erheblichen Reduzierung der Modellparameter führt.

Warum funktioniert dieser Mechanismus? Die Autoren stellen fest, dass bei der Analyse der Parameter in jeder Schicht von BERT festgestellt wurde, dass die Parameter in jeder Schicht weitgehend ähnlich sind. Daher können die Parameter direkt über die Schichten hinweg geteilt werden.

Model		Parameters	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg
ALBERT base E=768	all-shared	31M	88.6/81.5	79.2/76.6	82.0	90.6	63.3	79.8
	shared-attention	83M	89.9/82.7	80.0/77.2	84.0	91.4	67.7	81.6
	shared-FFN	57M	89.2/82.1	78.2/75.4	81.5	90.8	62.6	79.5
	not-shared	108M	90.4/83.2	80.4/77.6	84.5	92.8	68.2	82.3
ALBERT base E=128	all-shared	12M	89.3/82.3	80.0/77.1	82.0	90.3	64.0	80.1
	shared-attention	64M	89.9/82.8	80.7/77.9	83.4	91.9	67.6	81.7
	shared-FFN	38M	88.9/81.6	78.6/75.6	82.3	91.7	64.4	80.2
	not-shared	89M	89.9/82.8	80.3/77.3	83.2	91.5	67.9	81.6

Tabelle 5 : Vergleich Pamameter von ALBERT bei verschidenen E [42]

In Bezug auf die Trainingsaufgabe wurde die Sentence-order prediction (SOP) als Ersatz für NSP vorgeschlagen.

Die Autoren haben die Probleme von NSP bei der Trainingseffektivität einiger Modelle wie XLNet und RoBERTa analysiert und festgestellt, dass dies nicht nur die Vorhersage von Beziehungen zwischen Sätzen umfasst, sondern auch die Vorhersage von Themen. Die Vorhersage von Themen ist offensichtlich einfacher, und das Modell tendiert dazu, die Vorhersage anhand der thematischen Zusammenhänge zu treffen. Aus diesem Grund wurde die Trainingstask durch SOP (sentence order prediction) ersetzt, bei der die Vorhersage besteht, ob zwei Sätze in ihrer Reihenfolge vertauscht wurden. Experimente haben gezeigt, dass diese neue Aufgabe eine Verbesserung von 1 Punkt erzielt.

	Intrinsic Tasks			Downstream Tasks					
SP tasks	MLM	NSP	SOP	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg
None	54.9	52.4	53.3	88.6/81.5	78.1/75.3	81.5	89.9	61.7	79.0
NSP	54.5	90.5	52.0	88.4/81.5	77.2/74.6	81.6	**91.1**	62.3	79.2
SOP	54.0	78.9	86.5	**89.3/82.3**	**80.0/77.1**	**82.0**	90.3	**64.0**	**80.1**

Tabelle 6 : Vergleich Ergebnisse von ALBERT bei verschidenen Trainingsaufgaben [42]

Wie oben erwähnt, reduziert ALBERT durch die gemeinsame Nutzung von Parametern, die Faktorisierung der Einbettungen und die Einführung einer neuen Vortrainingsaufgabe signifikant die Anzahl der Parameter, verbessert die Trainingseffizienz und behält gleichzeitig eine hohe Modellleistung bei. Diese Merkmale haben dazu geführt, dass ALBERT in der Ingenieurwissenschaft weit

verbreitet eingesetzt wird.

3.2.1.4 DeBERTa（Decoding-enhanced BERT with disentangled attention）[43]

DeBERTa ist ein verbessertes Modell von BERT, das zwei neue Techniken verwendet, um das damals fortschrittlichste Vortrainingsmodell zu verbessern: die Decoupled Self-Attention-Mechanismus und den Enhanced Masked Decoder.

Model	CoLA Mcc	QQP Acc	MNLI-m/mm Acc	SST-2 Acc	STS-B Corr	QNLI Acc	RTE Acc	MRPC Acc	Avg.
BERT_{large}	60.6	91.3	86.6/-	93.2	90.0	92.3	70.4	88.0	84.05
RoBERTa_{large}	68.0	92.2	90.2/90.2	96.4	92.4	93.9	86.6	90.9	88.82
XLNet_{large}	69.0	92.3	90.8/90.8	**97.0**	92.5	94.9	85.9	90.8	89.15
$\text{ELECTRA}_{\text{large}}$	69.1	**92.4**	90.9/-	96.9	92.6	95.0	88.0	90.8	89.46
DeBERTa_{large}	**70.5**	92.3	**91.1/91.1**	96.8	**92.8**	**95.3**	**88.3**	**91.9**	**90.00**

Tabelle 7 : Vergleich Ergebnisse von verschidenen PLMs [43]

Die **Decoupled Self-Attention-Mechanismus** (entkoppelte Selbst-Aufmerksamkeit) ist ein Mechanismus, der in DeBERTa verwendet wird. In herkömmlichen Self-Attention-Mechanismen wird für jede Position in einer Eingabesequenz die Aufmerksamkeit auf alle anderen Positionen berechnet. Dieser Mechanismus geht davon aus, dass jede Position in der Sequenz gleich wichtig für die aktuelle Position ist. In der Praxis trifft diese Annahme jedoch oft nicht zu. In Aufgaben des semantischen Verstehens hat beispielsweise ein Wort in der Regel mehr Einfluss auf seine Bedeutung durch den umgebenden Kontext als durch seine Position in der Sequenz.

Um dieses Problem zu lösen, führt DeBERTa den Decoupled Self-Attention-Mechanismus ein. Dieser Mechanismus teilt die Repräsentation jeder Eingabeposition in zwei Teile auf: einen Teil, der den Einfluss dieser Position auf andere Positionen berechnet (Influence Score), und einen Teil, der angibt, wie sehr diese Position von anderen Positionen beeinflusst wird (Affected Score). Auf diese Weise kann dem Modell unterschiedliche Einfluss- und Betroffenheitsgewichte für jede Eingabeposition zugewiesen werden, was dem Modell ermöglicht, die Eingabesequenz flexibler zu verarbeiten.

Der Decoupled Self-Attention-Mechanismus verwendet separate Gewichtsmatrizen für die Berechnung der Influence Scores und Affected Scores, was zusätzliche Komplexität in den Modellparametern einführt. Jedoch verbessert dieser Mechanismus die Flexibilität und Leistungsfähigkeit des Modells deutlich.

Im **Enhanced Masked Decoder (EMD)** von DeBERTa wird die absolute Positionsinformation der Sequenz nach der letzten Transformer-Schicht in den Softmax-Layer eingespeist, bevor die Maskierungsvorhersage erfolgt. Dadurch wird der Decoder-Teil des DeBERTa-Modells verbessert. In Experimenten wurde festgestellt, dass EMD im Vergleich zur in BERT verwendeten Methode (direkte Zusammenführung der absoluten Positionsinformation in den Input) eine deutlich bessere Leistung erzielt.

3.2.1.5 ELECTRA（Efficiently Learning an Encoder that Classifies Token Replacements Accurately）[31]

Die Modellstruktur von ELECTRA sieht wie folgt aus:

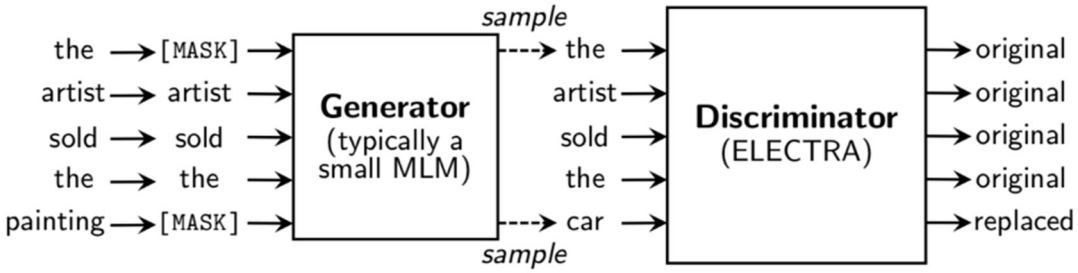


Abb.24 Struktur von ELECTRA[31]

Die Pretraining-Phase von ELECTRA umfasst zwei separate Modelle: den Generator und den Diskriminator.

Generator: Der Generator ist ein kleines Sprachmodell, dessen Aufgabe es ist, für eine gegebene Eingabesequenz mögliche Ersatzwörter zu generieren. Der Generator erhält zunächst eine Eingabesequenz, in der einige Wörter zufällig durch spezielle Maskierungssymbole (z. B. "[MASK]") ersetzt werden. Anschließend versucht der Generator vorherzusagen, welche Wörter maskiert wurden.

Diskriminator: Der Diskriminator ist ein größerer Transformer-Modell, dessen Aufgabe es ist, zu bestimmen, ob jedes Wort in der Eingabesequenz vom Generator gefälscht wurde. Der Diskriminator weiß nicht, welche Wörter vom Generator maskiert wurden, er weiß nur, dass der Generator an einigen Stellen Ersatz vorgenommen haben könnte. Daher besteht die Aufgabe des Diskriminators darin, diese Ersatzvorgänge zu identifizieren, anstatt die maskierten Wörter vorherzusagen (wie es im Fall von Bert der Fall ist). Diese neue Vortrainingsaufgabe wird als "Masked Token Detection" bezeichnet, und ihr Hauptvorteil besteht darin, dass sie alle Wörter in der Eingabesequenz nutzen kann, nicht nur die nicht maskierten Wörter. Dies ist auch eine wesentliche Innovation des ELECTRA-Modells.

In den Experimenten wurde beobachtet, dass die beste Leistung erzielt wurde, wenn die Größe des Generators etwa 1/4 bis 1/2 der Größe des Diskriminators entsprach. Der Grund dafür, wie die Autoren feststellen, liegt darin, dass ein zu starker Generator die Aufgabe des Diskriminators erschwert.[31]

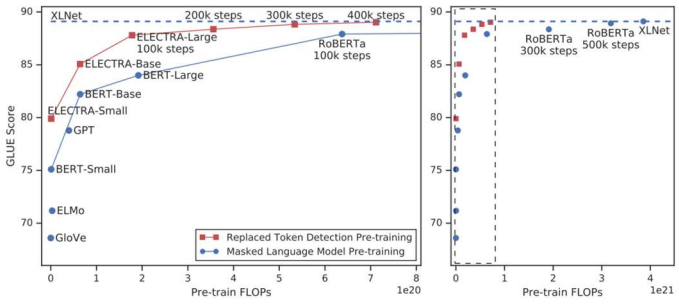


Abb.25 Trainingsgeschwindigkeit im Vergleich zu ELECTRA [31]

Aus dem Diagramm ist ersichtlich, dass das ELECTRA-Modell durch die Verwendung einer neuen Vortrainingsaufgabe und Änderungen in der Modellstruktur bei gleicher Rechenleistung eine bessere Leistung als die Bert- und RoBERTa-Modelle erzielt.

3.2.2 Decoder-basierte Modelle

Decoder-basierte Modelle konzentrieren sich hauptsächlich darauf, wie neue Daten aus extrahierten Informationen generiert werden können. Sie verwenden in der Regel die Decoder-Struktur des Transformers und sind gut darin, Aufgaben im Bereich der Textgenerierung zu bewältigen.

3.2.2.1 GPT-1（Generative Pre-Trained Transformer）

GPT-1 (Generative Pre-Trained Transformer) unterscheidet sich von den BERT-Modellen durch die Verwendung des Decoder-Teils des Transformers. GPT-1 wurde einige Monate vor BERT entwickelt. Beide Modelle nutzen den Transformer als Kernstruktur, jedoch verwendet GPT-1 eine generative, links-nach-rechts Ausrichtung für die Aufgabe des Pre-Trainings. Dadurch wird ein allgemeines Pre-Trained-Modell erstellt, das ähnlich wie BERT für die Feinabstimmung in verschiedenen Downstream-Aufgaben verwendet werden kann. Aus der Grafik geht hervor, dass bei der Berechnung der Aufmerksamkeit jedes Tokens nur die Wörter auf der linken Seite betrachtet werden können, im Gegensatz zu BERT, wo das gesamte Sequenzkontext berücksichtigt wird.

GPT-1 verwendet das Sprachmodell (Language Model) als Vortrainingsaufgabe.

Feinabstimmung

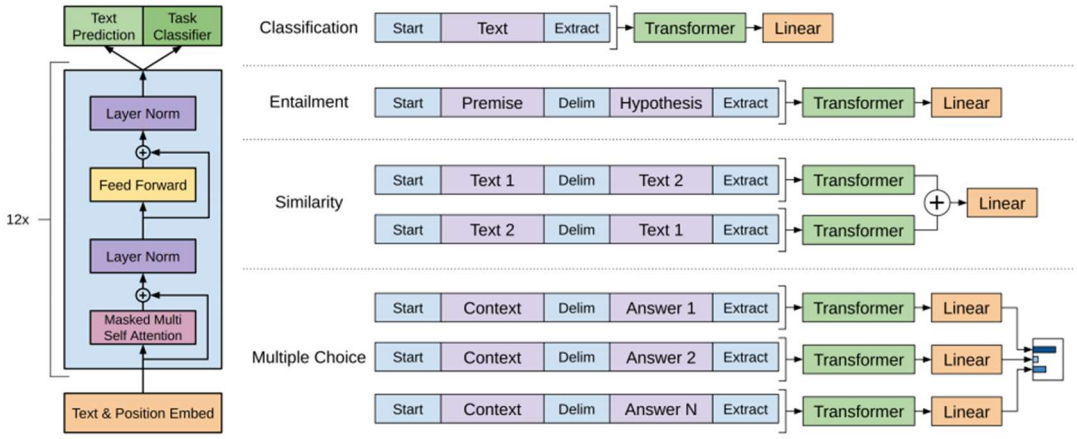


Abb.26 Feinabstimmungsphase GPT-1 [5]

Basierend auf dem Diagramm kann die Feinabstimmung in vier Kategorien unterteilt werden:

- Für Aufgaben der Textklassifikation werden Start- und Endmarkierungen vor und nach der Eingabesequenz hinzugefügt.
- Für Aufgaben der Satzrelation (z. B. Textentailment) werden neben den Start- und Endmarkierungen für die Prämisse (Premise) und Hypothese auch ein Trennzeichen (Delim) zwischen den beiden Sätzen hinzugefügt.
- Für Aufgaben der Textähnlichkeit wird aufgrund der Struktur des maskierten Selbstattention-Mechanismus unterschiedliche Ergebnisse für die Kombination von Text1+Text2 und Text2+Text1 erzeugt. Daher werden die beiden Sätze in umgekehrter Reihenfolge in das GPT-Modell eingegeben, die erzeugten Ergebnisse werden addiert und dann durch eine binäre Klassifikationsschicht ausgewertet.
- Für Frage-Antwort-Aufgaben (z. B. Multiple-Choice-Leseaufgaben, bei denen der Computer einen Text liest und anhand einer gegebenen Frage eine korrekte Option aus mehreren Optionen auswählt) erfolgt die Eingabe in Form von <Text, Frage, Option>. Wenn es n Optionen gibt, werden n Sätze einzeln zusammengesetzt. Jeder Satz wird durch das Modell geführt und dann mit einer vollständig verbundenen Schicht ausgewertet. Schließlich wird die Ausgabe der vollständig verbundenen Schicht verwendet, um die Vorhersage für n Optionen zu treffen.

GPT-1 erzielte damals herausragende Ergebnisse in neun NLP-Aufgaben, obwohl die Modellgröße und die Datenmenge vergleichsweise gering waren, was zur Entwicklung von GPT-2 führte.

3.2.2.2 GPT-2

GPT-2 baut auf GPT-1 auf und verwendet weiterhin den Transformer als Merkmalsextraktor sowie ein einseitiges Sprachmodell als Trainingsaufgabe. GPT-2

untersucht die Fähigkeit von Sprachmodellen für Zero-Shot-Lernen und zeigt die Ambitionen von OpenAI, im Bereich NLP eine allgemeine KI zu entwickeln. Die Hauptverbesserungen von GPT-2 liegen in den Daten und der Modellgröße[37]:

1 Höhere Qualität, breitere Abdeckung und größere Datenmenge

Um eine größere Menge an unüberwachten Trainingsdaten zu erhalten, verwendet GPT-2 8 Millionen Webseiten aus dem Internet, bekannt als WebText, für das Training des Sprachmodells. Der Vorteil von Internetseiten liegt darin, dass sie ein breites Spektrum an Themen abdecken. Dadurch wird ein Sprachmodell mit hoher Allgemeinheit trainiert, das nahezu jeden Bereich abdecken kann und somit für beliebige Aufgaben in verschiedenen Bereichen verwendet werden kann. Neben der großen Datenmenge und der hohen Allgemeinheit ist auch die Datenqualität wichtig. Hochwertige Daten enthalten bessere Sprache und menschliches Wissen. Daher hat GPT-2 auch eine Qualitätsfilterung durchgeführt, um hochwertige Webseiteninhalte auszuwählen.

2 Ein riesigeres Transformer-Modell mit 1,5 Milliarden Parametern

GPT-2 als verbesserte Version von GPT-1 hat die Anzahl der Schichten auf 48 und die Parameter auf 1,5 Milliarden erhöht. Das Ziel war es, mit mehr Trainingsdaten das Modell vorzubereiten. Mit einem größeren Modell und mehr Parametern bedeutet dies eine höhere Modellkapazität.

GPT-2 trainierte ein größeres Sprachmodell mit einer großen Menge hochwertiger Daten in der ersten Phase. Um die Effektivität und Allgemeinheit des trainierten Modells zu demonstrieren, wurde in der zweiten Phase keine überwachte Feinabstimmung durchgeführt, sondern es wurde direkt verwendet.

Bei der Durchführung von unbeaufsichtigten Aufgaben im Zusammenhang mit den untergeordneten Aufgaben generiert GPT-2 Ausgaben, die identisch mit den Ausgaben des Sprachmodells sind. Es werden einzelne Wörter ausgegeben, die dann zu den gewünschten Ergebnissen der Aufgaben verbunden werden. Um beispielsweise eine Zusammenfassung zu erstellen, wird ein Leitfaden wie "TL:DR" als Eingabe hinzugefügt, um GPT-2 zu einer korrekten Vorhersage des Ziels zu leiten.

Durch GPT-2 wurde deutlich, dass die Kombination von größerer Modellkapazität und hochwertigen Daten zu erstaunlichen Ergebnissen führen kann.

3.2.2.3 GPT-3[44]

Obwohl GPT-2 im Bereich der Innovation mit seiner Zero-Shot-Fähigkeit bemerkenswert war, konnte es aufgrund seiner durchschnittlichen Leistung keine große Wirkung in der Industrie erzielen. GPT-3 wurde entwickelt, um die Leistungsprobleme anzugehen. GPT-3 verfolgt nicht mehr das Ziel, ein Modell zu schaffen, das ohne jegliche Beispiele gut abschneiden kann, sondern strebt nach einer Lernmethode, die der menschlichen Art des Lernens ähnelt. Es soll möglich sein, mit nur einer sehr geringen Anzahl von Beispielen eine Aufgabe zu beherrschen. Daher lautet das Motto von GPT-3: "Language Models are Few-Shot Learners".

Im Falle von GPT-3 bedeutet Few-Shot Learning jedoch nicht, dass nur wenige Beispiele für das Feintuning auf einer spezifischen Aufgabe verwendet werden. Aufgrund der enormen Anzahl von Parametern in GPT-3 wäre selbst das Feintuning zu kostspielig.

Modellarchitektur:

Bei der Modellarchitektur setzt GPT-3 weiterhin auf die GPT-Struktur, führt jedoch das Sparse Attention Modul aus Sparse Transformer[45] ein. Die Verwendung von Sparse Attention bietet folgende Vorteile:

Reduzierung der Berechnungskomplexität der Aufmerksamkeitsschicht, was Speicherplatz und Zeit spart und längere Eingabesequenzen verarbeiten kann.

Es hat die Eigenschaft, "lokale Dichte und entfernte Sparsamkeit" aufzuweisen, das heißt, es wird mehr Aufmerksamkeit auf nahegelegenen Kontext gerichtet und weniger auf entfernten Kontext.

Weitere Informationen zur Sparse Attention finden Sie in der Veröffentlichung "Generating Long Sequences with Sparse Transformers"[45].

Letztendlich hat GPT-3 im Trainingsprozess Modelle unterschiedlicher Größen erreicht. Das größte Modell, GPT-3, hat 175 Milliarden Parameter.

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or “GPT-3”	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

Tabelle 8 : Vergleich Parameter von GPT-3 Modellen[44]

GPT-3 bietet drei verschiedene Ansätze zur Bewertung und Vorhersage von Aufgaben im Kontext von Downstream-Aufgaben:

Zero-Shot: Es wird nur die natürliche Sprachbeschreibung der aktuellen Aufgabe verwendet.

One-Shot: Zusätzlich zur natürlichen Sprachbeschreibung der aktuellen Aufgabe wird ein einfaches Eingabe-Ausgabe-Beispiel bereitgestellt.

Few-Shot: Zusätzlich zur natürlichen Sprachbeschreibung der aktuellen Aufgabe werden einige einfache Eingabe-Ausgabe-Beispiele bereitgestellt.

Dies ist auch der erste Ansatz, bei dem das Konzept des "in-context learning" eingeführt wird. Obwohl sowohl "Feinabstimmung" als auch "in-context learning" eine gewisse Menge an überwachten Daten erfordern, gibt es Unterschiede:

Bei der "Feinabstimmung" werden die Modellparameter anhand der markierten Daten

aktualisiert, während beim "in-context learning" bei Verwendung der markierten Daten kein Gradientenrückfluss erfolgt und die Modellparameter nicht aktualisiert werden.

"in-context learning" erfordert viel weniger Daten als herkömmliche "Feinabstimmung".

In umfangreichen Experimenten mit verschiedenen Downstream-Aufgaben wurde festgestellt, dass der Few-Shot-Ansatz die beste Leistung erzielt, gefolgt vom One-Shot-Ansatz, während der Zero-Shot-Ansatz die schlechteste Leistung zeigt.

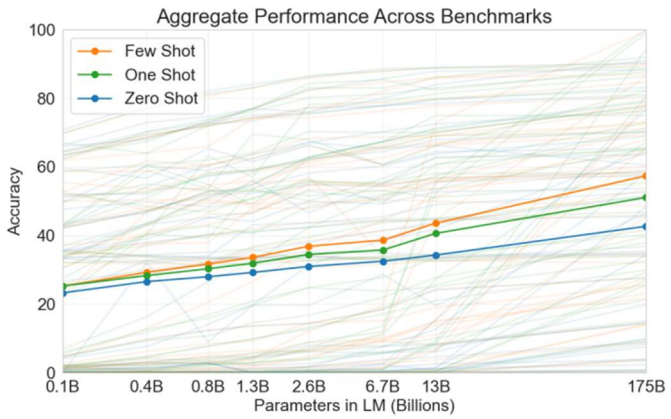


Abb.27 Benchmark in Bezug auf Modellgröße [44]

Im Diagramm sind auf der X-Achse die Modellparameter und auf der Y-Achse die Genauigkeit der Aufgaben dargestellt. Die grauen Linien repräsentieren eine Vielzahl von verschiedenen Downstream-Aufgaben, während die orangefarbenen/grünen/blauen Linien den Durchschnitt der Aufgabengenauigkeit darstellen.

Trainingsdaten

Aufgrund der Skalierung des GPT-3-Modells in Bezug auf die Modellgröße mussten auch die Trainingsdaten entsprechend erweitert werden, um das volle Potenzial des Modells nutzen zu können.

GPT-3 verwendet mehrere Datensätze, wobei der größte Datensatz CommonCrawl ist. Die Rohdaten umfassen etwa 45 TB, und nach der Bereinigung bleibt eine Datenmenge von rund 570 GB übrig. Insgesamt beträgt das Trainingsdatenvolumen etwa 750 GB.[44]

Experimentelle Analyse

GPT-3 widmet einen Großteil seiner Präsentation den experimentellen Ergebnissen und Analysen in verschiedenen NLP-Aufgaben. Aufgrund des begrenzten Umfangs dieser Zusammenfassung empfehle ich den Lesern, das Originaldokument für eine detailliertere Analyse zu lesen.

Einschränkungen von GPT-3

- Obwohl GPT-3 beeindruckende Ergebnisse erzielt hat, werden in der Arbeit auch einige Einschränkungen objektiv analysiert:
- Wenn die generierte Textlänge lang ist, können immer noch Probleme auftreten, wie z. B. wiederholte Sätze, Widersprüche oder schlechte logische Zusammenhänge.
- Begrenzungen des Modells und der Struktur: Für bestimmte Aufgaben, wie Lückentext-Aufgaben, gibt es gewisse Einschränkungen bei der Verwendung eines rein autoregressiven Sprachmodells. In solchen Fällen kann eine bessere Berücksichtigung des Kontextes sowohl vor als auch nach der Lücke zu besseren Ergebnissen führen.
- Die Effektivität und Nutzung der Stichproben ist relativ niedrig im Vergleich zu den Kosten, die für das menschliche Lernen erforderlich sind. Dies ist ein wichtiges Thema für zukünftige Forschungen im Bereich der Künstlichen Intelligenz.
- Es gibt eine gewisse Unsicherheit darüber, ob das Modell tatsächlich "lernt" oder nur "erinnert". Obwohl die Autoren natürlich eine Lernleistung des Modells anstreben, ist es schwierig zu bestimmen, wie es bei solch großen Datensätzen wirklich funktioniert.
- Die Kosten für das Training und den Einsatz von GPT-3 sind sehr hoch.
- GPT-3 ist wie viele andere Deep-Learning-Modelle nicht erklärbar. Es ist nicht möglich, zu verstehen, wie das Modell interne Entscheidungen trifft.
- Das endgültige Modellergebnis hängt stark von den Trainingsdaten ab, was zu verschiedenen "Vorurteilen" des Modells führen kann.[44]

3.2.3 Basierend auf dem Encoder-Decoder-Modell

Das Encoder-Decoder-Modell kombiniert die Strukturen des Encoders und Decoders und ist in der Lage, Informationen aus den Eingabedaten zu extrahieren und basierend auf diesen Informationen neue Daten zu generieren. Diese Art von Modellen eignet sich für verschiedene Aufgaben, bei denen Text verstanden und generiert werden muss, wie maschinelle Übersetzung, Textzusammenfassung, usw.

3.2.3.1 T5 (Transfer Text-to-Text Transformer)

T5 zielt darauf ab, einen einheitlichen Modellrahmen vorzuschlagen, bei dem alle Arten von NLP-Aufgaben als Text-to-Text-Aufgaben betrachtet werden, bei denen sowohl die Eingabe als auch die Ausgabe Text sind. Dadurch können verschiedene Modellstrukturen, Pretraining-Ziele und nicht gekennzeichnete Datensätze leicht in Bezug auf Aufgaben wie Leseverständnis, Zusammenfassungsgenerierung, Textklassifikation usw. bewertet werden.

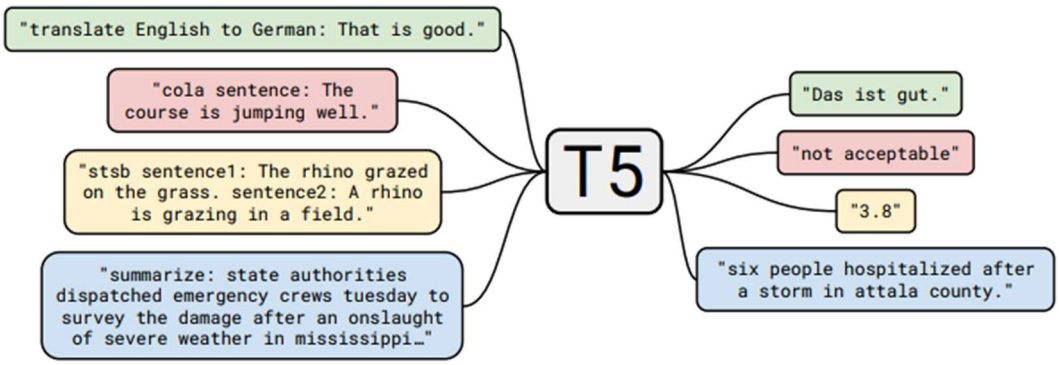


Abb.28 Konvertierung Aufgabe in Text-to-Text-Aufgaben [30]

Wie in der Abbildung dargestellt, konvertiert das T5-Modell Aufgaben wie Übersetzung, Klassifizierung, Regression, Zusammenfassungsgenerierung usw. in einheitliche Text-to-Text-Aufgaben. Dadurch können diese Aufgaben während des Trainings (Pre-Training und Feinabstimmung) dieselbe Ziel-Funktion verwenden und während der Testphase denselben Dekodierungsprozess verwenden.

Datensatz C4

Die Autoren filterten den öffentlich verfügbaren Webdatensatz Common Crawl, um Duplikate, minderwertige Texte, textähnlichem Code usw. zu entfernen. Schließlich wurde nur englischer Text beibehalten, um den Colossal Clean Crawled Corpus (C4) zu erstellen.

Eingabe-Ausgabe-Format

Bei der Feinabstimmung des Modells auf den verschiedenen Aufgaben wird jedem Eingabebeispiel ein präfixabhängiger Präfix hinzugefügt, um dem Modell mitzuteilen, um welche Art von Aufgabe es sich handelt.

- Übersetzung-Präfix: translate English to German:
- Klassifizierung-Präfix: cola sentence:
- Zusammenfassung-Präfix: summarize:

usw. Die Autoren stellten fest, dass verschiedene Präfixe nur begrenzten Einfluss auf das Modell haben, daher wurden keine umfangreichen Experimente durchgeführt, um verschiedene Präfixe zu vergleichen und auszuwählen.

Experimente

Die Autoren stellten zunächst ein Baseline-Modell auf und änderten dann jedes Mal einen Aspekt des Modells, um Vergleichsexperimente unter Verwendung der Methode der kontrollierten Variablen durchzuführen.

Architektur:

Zunächst verglichen die Autoren verschiedene Modellarchitekturen (Transformer) in

den vortrainierten Modellen. Die wichtigsten Modellarchitekturen lassen sich in folgende drei Kategorien einteilen.

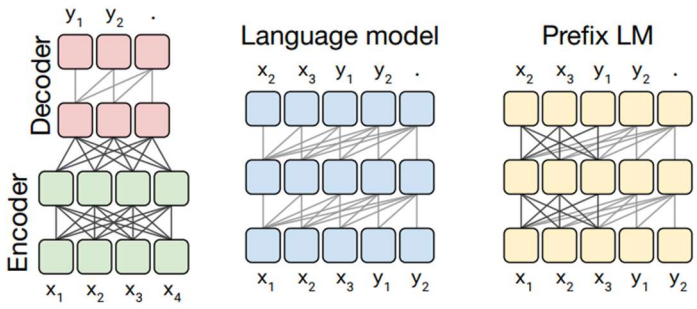


Abb.29 Drei Transformer Modellarchitekturen [30]

1 **Encoder-Decoder-Modelle**: In diesen Modellen kann der Encoder den gesamten Eingabesequenz sehen, während der Decoder nur den linken Teil der Ausgabe des Decoders sieht.

2 **Decoder-Modelle**: Diese Modelle ähneln dem Decoder-Teil des Transformer-Modells. Ein bekanntes Beispiel für diese Architektur ist die GPT-Serie.

3 **Prefix LM** (Language Model)-Modelle: Diese Modelle sind eine Kombination aus Encoder und Decoder. Ein Teil der Eingabesequenz, ähnlich wie beim Encoder, kann die gesamte Sequenz sehen, während ein anderer Teil, ähnlich wie beim Decoder, nur den vorherigen Teil der Sequenz sehen kann.

Alle diese Architekturen bestehen aus Transformern, und durch Maskenoperationen im Aufmerksamkeitsmechanismus entstehen diese verschiedenen Transformationen.

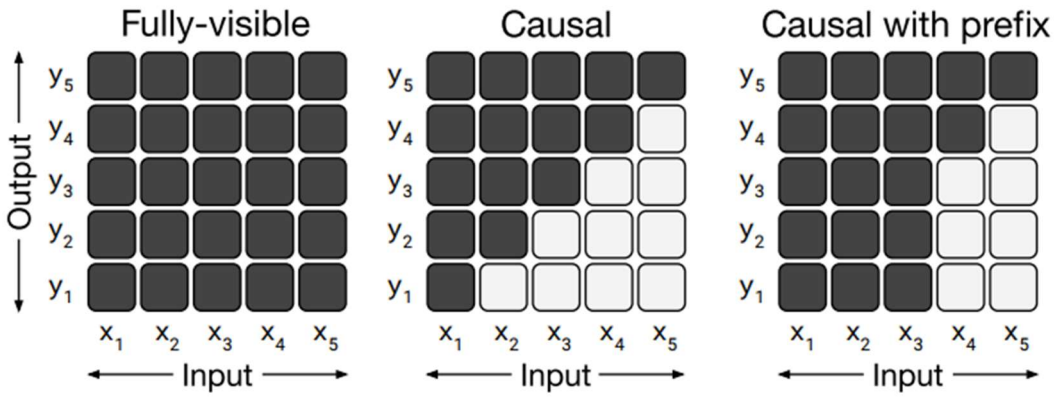


Abb.30 Maskenoperationen im Aufmerksamkeitsmechanismus [30]

Durch Experimente haben die Autoren festgestellt, dass das Encoder-Decoder-Modell in der vorgeschlagenen Text-to-Text-Architektur die beste Leistung erzielt. Daher haben sie es als T5-Modell festgelegt.

Im Anschluss wurden verschiedene Vortrainingsziele untersucht und verglichen. Insgesamt wurden vier Runden durchgeführt.

In der **ersten Runde** wurden drei Arten von selbstüberwachten Vortrainingsmethoden verglichen:

- **Sprachmodellierung**: ähnlich wie bei GPT-2, bei dem die Vorhersage von links nach rechts erfolgt.
- **BERT-Style**: ähnlich wie bei BERT, bei dem ein Teil der Eingabe zerstört und dann wiederhergestellt wird.
- **Deshuffling**: Hierbei wird der Text durcheinander gebracht und dann wieder in die richtige Reihenfolge gebracht.

Es stellte sich heraus, dass der BERT-Style die beste Leistung erzielte, daher wurde er für den weiteren Vergleich ausgewählt.

In der **zweiten Runde** wurden drei verschiedene Strategien zur Zerstörung eines Teils des Textes verglichen:

- Maskierung: Ähnlich wie bei den meisten aktuellen Modellen, bei denen die zerstörten Token durch spezielle Symbole wie [M] ersetzt werden.
- Replace Span: Hier werden benachbarte Maskierungssymbole [M] zu einem einzelnen speziellen Symbol zusammengefasst, und jede Zerstörungsstelle wird durch ein spezielles Symbol ersetzt, um die Berechnungseffizienz zu verbessern.
- Drop: Hierbei werden die zerstörten Zeichen einfach zufällig entfernt, ohne sie durch spezielle Symbole zu ersetzen.

Die Replace-Span-Methode wurde als Gewinner ausgewählt, und ähnliche Ansätze wie SpanBERT bestätigten deren Wirksamkeit.

In der **dritten Runde** wurden verschiedene Zerstörungsraten für den Text untersucht. Es wurden vier Werte getestet: 10%, 15%, 25% und 50%. Es stellte sich heraus, dass eine Zerstörungsrate von 15% äußerst effektiv war, ähnlich wie bei BERT.

In der **vierten Runde** wurde die Auswahl der Länge des zu zerstörenden Textabschnitts untersucht. Es wurden die Werte 2, 3, 5 und 10 getestet, wobei sich eine Länge von 3 als die beste erwies.

Nach diesen vier Runden wurde das vollständige T5-Modell entwickelt, zusammen mit der entsprechenden Trainingsmethode.

Neben der Festlegung der optimalen Modellstruktur und Trainingsmethode wurden auch andere Aspekte verglichen, wie die Notwendigkeit der Datenbereinigung, die Bedeutung der Datenmenge sowie der Einfluss der Modellgröße und Trainingszeit auf die Leistung des Modells.

Die T5-Studie fasst die verschiedenen Trainingstechniken im aktuellen NLP-Bereich zusammen und vergleicht sie fair. Sie analysiert den tatsächlichen Einfluss dieser Trainingstechniken auf die Leistung des Modells, um ein gutes Vortrainingsmodell mit geeigneten Techniken zu erstellen.

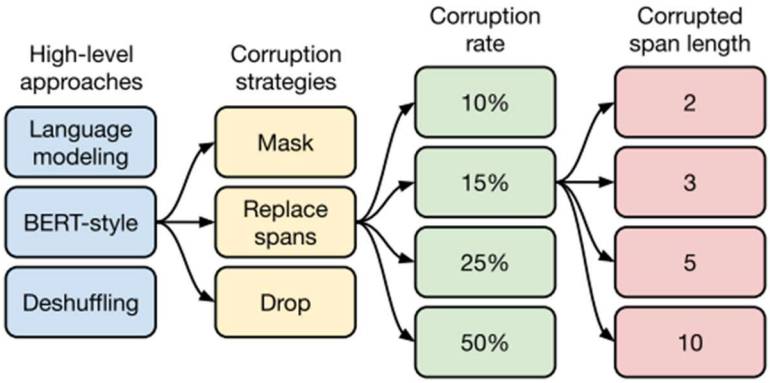


Abb.31 Auswahl der besten Strategie in vier Runden [30]

3.3 Zusammenfassung

In dieser Phase der Entwicklung, die sich auf die zwei Jahre nach der Einführung von BERT konzentrierte, wurden viele namhafte Forschungsinstitute und Unternehmen intensiv mit den Transformer-Modellen im Bereich NLP beschäftigt. Vom BERT zum RoBERTa und weiter zum ALBERT und Google T5 markieren diese Modelle den fortschreitenden technischen Fortschritt und die Vertiefung im Bereich NLP.

In dieser Phase wurde nach dem besten Weg für die Modellentwicklung gesucht. Dies umfasste nicht nur die Auswahl der Modellarchitektur und Trainingsstrategie, sondern auch die effektivere Nutzung großer Mengen an unbeschrifteten Textdaten, das bessere Erlernen und Verständnis von Sprachwissen sowie die effektive Anwendung dieses erlernten Sprachwissens auf Aufgaben im Bereich NLP. Die Forschung in dieser Zeit hat nicht nur die technologische Entwicklung im Bereich NLP vorangetrieben, sondern uns auch wertvolle Erfahrungen und Erkenntnisse für das Verständnis und den Einsatz von vorab trainierten Modellen geliefert.

Der eigentliche Zweck von vortrainierten Modellen besteht darin, dass sie mithilfe von selbstüberwachtem Lernen Sprachwissen aus unbeschriftetem Freitext extrahieren und erlernen, basierend auf der Grundstruktur des Transformers. Dieses Sprachwissen wird in Form von Parametern in die Transformer-Struktur eingebettet und steht dann für nachfolgende Aufgaben zur Verfügung.

Die verschiedenen Arten von vortrainierten Modellen zielen alle darauf ab, dieses Ziel zu erreichen. Ihre Unterschiede liegen in der Modellarchitektur, den Lernaufgaben und anderen Komponenten. Obwohl die Ergebnisse variieren können, besteht das übergeordnete Ziel darin, Sprachwissen aus dem Freitext zu lernen. Es ist erwähnenswert, dass neben dem Freitext auch neue Daten und Kenntnisse wie strukturiertes Wissen, multimodale Daten und mehrsprachige Daten eingeführt werden können, um die sprachliche Verständnisfähigkeit des Modells zu verbessern oder spezifische Aufgaben anzugehen.

RoBERTa kann als eine tiefer trainierte Version von BERT betrachtet werden, während

ALBERT/Google T5 als Versionen von RoBERTa mit weiter erhöhter Modellkomplexität angesehen werden können. Diese drei Kategorien von Modellen repräsentieren wahrscheinlich die wichtigsten technischen Fortschritte seit der Einführung von BERT. Obwohl die neuen vorab trainierten Modelle eine signifikante Verbesserung bei der Lösung von NLP-Aufgaben im Vergleich zu BERT zeigen, kann diese Verbesserung im Allgemeinen auf den Einsatz von mehr hochwertigen Daten, eine erhöhte Modellgröße, tiefere Modelltrainings und eine Erhöhung der Schwierigkeit der Trainingsaufgaben zurückgeführt werden.

Dies verdeutlicht die Wirksamkeit bei großen Datenmengen und großen Modellen, da diese direkte und effektive Methode noch erhebliches Potenzial bietet. Obwohl dies bedeutet, dass das Training ausgezeichneter vorab trainierter Modelle sehr kostspielig ist und nicht von allen Forschern durchgeführt werden kann, ist dies ein effektiver Weg, um die Leistung von Modellen kontinuierlich zu verbessern und bessere Ergebnisse in verschiedenen Anwendungen zu erzielen.

Die Einführung von GPT-3 im Jahr 2020 bestätigte den Erfolg des Wegs mit großen Datenmengen und großen Modellen. GPT-3 hat unser Verständnis von natürlicher Sprachverarbeitung auf eine neue Ebene gebracht und mit seiner Komplexität und riesigen Modellgröße einen neuen Maßstab für KI-Technologien gesetzt. Wie man diese großen Modelle effektiv einsetzen kann und wie man mit den Herausforderungen umgeht, die sie möglicherweise mit sich bringen, sind auch heute noch wichtige Themen im Bereich NLP.

Im nächsten Kapitel werden wir uns auf die Erforschung und das Verständnis von Large Language Models (LLM) konzentrieren.

Kapitel 4 Ära der Großen Modelle

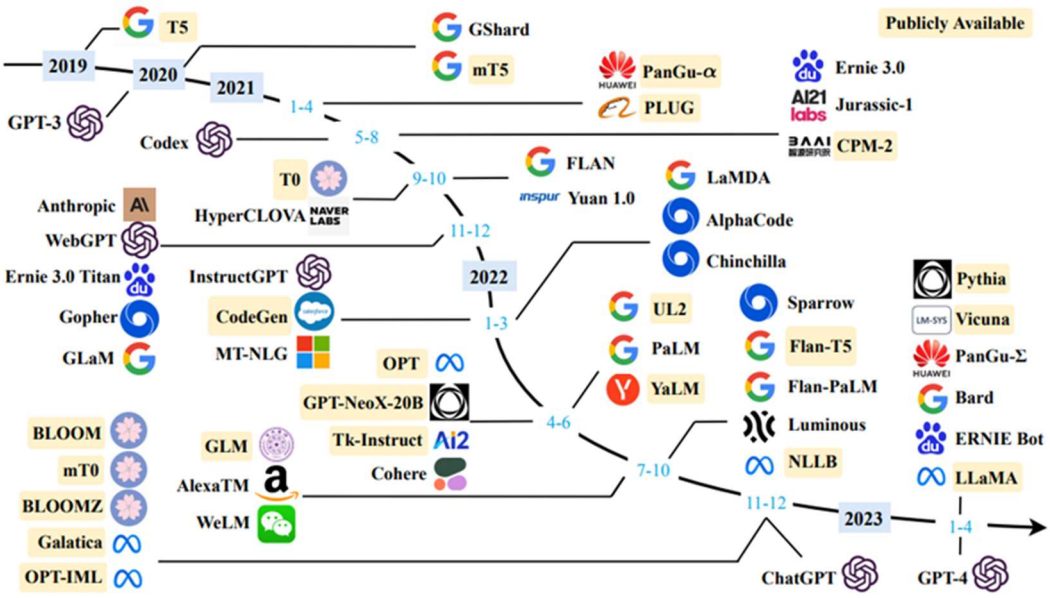


Abb.32 Entwicklung der großen Modelle [46]

In den letzten Jahren hat das Feld der Natural Language Processing (NLP) eine revolutionäre Veränderung erlebt. GPT-3, ein großes Sprachmodell, entwickelt von OpenAI, hat diese Transformation angeführt. Mit seinen beeindruckenden 175 Milliarden Parametern wurde GPT-3 zum damals größten Sprachmodell der Welt und markierte den Eintritt des NLP-Feldes in das Zeitalter großer Modelle.

Normalerweise bezieht sich der Begriff "großes Sprachmodell" (LLM) auf Sprachmodelle mit mehreren Billionen (oder mehr) Parametern.[46] Mit dem Aufkommen und der Entwicklung großer Sprachmodelle (Large Language Models, LLM) betreten wir eine neue Ära, über die dieses Kapitel diskutieren wird, und wir werden auch die möglichen Zukunftsaussichten von LLM erkunden.

4.1 Training von großen Modellen

Das Training großer Sprachmodelle ist eine äußerst komplexe und anspruchsvolle Aufgabe. Es erfordert erhebliche Rechenressourcen wie Hochleistungs-GPUs oder TPUs, umfangreiche Trainingsdaten und ein hohes Maß an fachlichem Know-how.

- Ressourcenanforderungen: Das Training großer Modelle erfordert erhebliche Rechenressourcen, möglicherweise Hunderte oder Tausende von GPUs oder TPUs sowie viel Speicherplatz. Darüber hinaus wird eine hohe Netzwerkbandbreite benötigt, um die Synchronisierung der Modellparameter auf den verschiedenen Knotenpunkten während des verteilten Trainings zu gewährleisten.

- Speicheranforderungen: Die Anzahl der Parameter in großen Modellen ist so groß, dass sie nicht vollständig in den Speicher einer einzelnen GPU oder TPU passen. Daher müssen spezielle verteilte Trainingsstrategien wie Modellparallelisierung angewendet werden, bei denen verschiedene Teile des Modells auf verschiedenen Geräten gespeichert werden.
- Trainingsdaten: Große Modelle erfordern eine große Menge an Trainingsdaten, um gute Leistung zu erzielen. Die Beschaffung und Speicherung dieser Daten erfordert jedoch viel Zeit und Ressourcen und es müssen auch Fragen des Datenschutzes und des Urheberrechts berücksichtigt werden.
- Trainingszeit: Das Training großer Modelle dauert in der Regel sehr lange, möglicherweise mehrere Tage, Wochen oder sogar Monate. Während dieser Zeit ist eine kontinuierliche Überwachung erforderlich, um die Stabilität des Trainingsprozesses sicherzustellen, und es müssen flexible Anpassungen der Lernrate, Optimierungsalgorithmen und anderer Parameter vorgenommen werden, um eine Konvergenz des Modells zu gewährleisten.[47]

	Model	Release Time	Size (B)	Base Model	Adaptation IT	Adaptation RLHF	Pre-train Data Scale	Latest Data Timestamp	Hardware (GPUs / TPUs)	Training Time	Evaluation ICL	Evaluation CoT
Publicly Available	T5	Oct-2019	11	-	-	-	1T tokens	Apr-2019	1024 TPU v3	-	✓	-
	mT5	Oct-2020	13	-	-	-	1T tokens	-	-	-	✓	-
	PanGu-α	Apr-2021	13*	-	-	-	1.1TB	-	2048 Ascend 910	-	✓	-
	CPM-2	Jun-2021	198	-	-	-	2.6TB	-	-	-	-	-
	T0	Oct-2021	11	T5	✓	-	-	-	512 TPU v3	27 h	✓	-
	CodeGen	Mar-2022	16	-	-	-	577B tokens	-	-	-	✓	-
	GPT-NeoX-20B	Apr-2022	20	-	-	-	825GB	-	96 40G A100	-	✓	-
	Tk-Instruct	Apr-2022	11	T5	✓	-	-	-	256 TPU v3	4 h	✓	-
	UL2	May-2022	20	-	-	-	1T tokens	Apr-2019	512 TPU v4	-	✓	✓
	OPT	May-2022	175	-	-	-	180B tokens	-	992 80G A100	-	✓	-
	NLLB	Jul-2022	54.5	-	-	-	-	-	-	-	✓	-
	GLM	Oct-2022	130	-	-	-	400B tokens	-	768 40G A100	60 d	✓	-
	Flan-T5	Oct-2022	11	T5	✓	-	-	-	-	-	✓	✓
	BLOOM	Nov-2022	176	-	-	-	366B tokens	-	384 80G A100	105 d	✓	-
	mT0	Nov-2022	13	mT5	✓	-	-	-	-	-	✓	-
	Galactica	Nov-2022	120	-	-	-	106B tokens	-	-	-	✓	✓
	BLOOMZ	Nov-2022	176	BLOOM	✓	-	-	-	-	-	✓	-
	OPT-IML	Dec-2022	175	OPT	✓	-	-	-	128 40G A100	-	✓	✓
	LLaMA	Feb-2023	65	-	-	-	1.4T tokens	-	2048 80G A100	21 d	✓	-
	Pythia	Apr-2023	12	-	-	-	300B tokens	-	256 40G A100	-	✓	-
Closed Source	GPT-3	May-2020	175	-	-	-	300B tokens	-	-	-	✓	-
	GShard	Jun-2020	600	-	-	-	1T tokens	-	2048 TPU v3	4 d	-	-
	Codex	Jul-2021	12	GPT-3	-	-	100B tokens	May-2020	-	-	✓	-
	ERNIE 3.0	Jul-2021	10	-	-	-	375B tokens	-	384 V100	-	✓	-
	Jurassic-1	Aug-2021	178	-	-	-	300B tokens	-	800 GPU	-	✓	-
	HyperCLOVA	Sep-2021	82	-	-	-	300B tokens	-	1024 A100	13.4 d	✓	-
	FLAN	Sep-2021	137	LaMDA	✓	-	-	-	128 TPU v3	60 h	✓	-
	Yuan 1.0	Oct-2021	245	-	-	-	180B tokens	-	2128 GPU	-	✓	-
	Anthropic	Dec-2021	52	-	-	-	400B tokens	-	-	-	✓	-
	WebGPT	Dec-2021	175	GPT-3	-	✓	-	-	-	-	✓	-
	Gopher	Dec-2021	280	-	-	-	300B tokens	-	4096 TPU v3	920 h	✓	-
	ERNIE 3.0 Titan	Dec-2021	260	-	-	-	300B tokens	-	2048 V100	28 d	✓	-
	GLaM	Dec-2021	1200	-	-	-	280B tokens	-	1024 TPU v4	574 h	✓	-
	LaMDA	Jan-2022	137	-	-	-	2.81T tokens	-	1024 TPU v3	57.7 d	-	-
	MT-NLG	Jan-2022	530	-	-	-	270B tokens	-	4480 80G A100	-	✓	-
	AlphaCode	Feb-2022	41	-	-	-	967B tokens	Jul-2021	-	-	-	-
	InstructGPT	Mar-2022	175	GPT-3	✓	✓	-	-	-	-	✓	-
	Chinchilla	Mar-2022	70	-	-	-	1.4T tokens	-	-	-	✓	-
	PaLM	Apr-2022	540	-	-	-	780B tokens	-	6144 TPU v4	-	✓	✓
	AlexaTM	Aug-2022	20	-	-	-	1.3T tokens	-	128 A100	120 d	✓	✓
	Sparrow	Sep-2022	70	-	-	✓	-	-	64 TPU v3	-	✓	-
	WeLM	Sep-2022	10	-	-	-	300B tokens	-	128 A100 40G	24 d	✓	-
	U-PaLM	Oct-2022	540	PaLM	-	-	-	-	512 TPU v4	5 d	✓	✓
	Flan-PaLM	Oct-2022	540	PaLM	✓	-	-	-	512 TPU v4	37 h	✓	✓
	Flan-U-PaLM	Oct-2022	540	U-PaLM	✓	-	-	-	-	-	✓	✓
	GPT-4	Mar-2023	-	-	✓	✓	-	-	-	-	✓	✓
	PanGu-Σ	Mar-2023	1085	PanGu-α	-	-	329B tokens	-	512 Ascend 910	100 d	✓	-

Tabelle 9 : Trainingsdaten von großen Modellen[46]

4.1.1 Modellarchitektur

In den frühen Entwicklungsphasen vortrainierter Modelle konvergierten die technischen Frameworks zu zwei verschiedenen Ansätzen: dem BERT-Modus und dem GPT-Modus. Obwohl der BERT-Ansatz allgemein beliebter war, stellte man mit fortschreitender Technologieentwicklung fest, dass die derzeit größten LLM-Modelle fast alle auf dem Decoder-Modus basieren, ähnlich wie GPT-3. Beispiele dafür sind GPT-3, PaLM[47], Gopher[48], Chinchilla[49], LaMDA[50] und andere. Dies könnte hauptsächlich auf zwei Gründe zurückzuführen sein:

Erstens hat das T5-Modell von Google eine formale Einheitlichkeit in der äußeren Erscheinungsform von Aufgaben des natürlichen Sprachverständnisses und der natürlichen Sprachgenerierung erreicht. Im T5-Modell haben Aufgaben des natürlichen Sprachverständnisses die gleiche Eingabe-Ausgabe-Form wie Generierungsaufgaben. Dies bedeutet, dass Generierungsaufgaben mit Aufgaben des natürlichen Sprachverständnisses kompatibel sein können, umgekehrt jedoch nicht so leicht. Der Vorteil dabei ist, dass ein und dasselbe LLM-Generierungsmodell nahezu alle NLP-Probleme lösen kann. Wenn hingegen der BERT-Ansatz verwendet wird, kann dieses LLM-Modell Generierungsaufgaben nicht gut bewältigen. Daher tendiert man eher dazu, Generierungsmodelle zu verwenden.

Der zweite Grund liegt darin, dass der GPT-Modus erforderlich ist, um Aufgaben mit Null-Beispiel-Hinweisen (zero shot prompting) oder wenigen Beispielen (few shot prompting) effektiv zu lösen. Es wurde in früheren Studien [51] gezeigt, dass der BERT-Ansatz bei der Feinabstimmung von Aufgaben im Vergleich zum GPT-Modus überlegen ist. Wenn es jedoch darum geht, Aufgaben mit zero-shot/few-shot zu lösen, ist der GPT-Modus dem BERT-Modus überlegen. Dies deutet darauf hin, dass Generierungsmodelle besser in der Lage sind, Aufgaben im Stil von Zero-Shot/Few-Shot-Prompting zu bewältigen, während der BERT-Modus von Natur aus Nachteile in dieser Art des Vorgehens hat.

Aus der folgenden Tabelle ist ebenfalls ersichtlich, dass derzeit die meisten großen Sprachmodelle auf dem gleichen Decoder-Modus wie die GPT-Serie basieren.

Model	Category	Size	Normalization	PE	Activation	Bias	#L	#H	d_{model}	MCL
GPT3	Causal decoder	175B	Pre Layer Norm	Learned	GeLU	✓	96	96	12288	2048
PanGU- α	Causal decoder	207B	Pre Layer Norm	Learned	GeLU	✓	64	128	16384	1024
OPT	Causal decoder	175B	Pre Layer Norm	Learned	ReLU	✓	96	96	12288	2048
PaLM	Causal decoder	540B	Pre Layer Norm	RoPE	SwiGLU	×	118	48	18432	2048
BLOOM	Causal decoder	176B	Pre Layer Norm	ALiBi	GeLU	✓	70	112	14336	2048
MT-NLG	Causal decoder	530B	-	-	-	-	105	128	20480	2048
Gopher	Causal decoder	280B	Pre RMS Norm	Relative	-	-	80	128	16384	2048
Chinchilla	Causal decoder	70B	Pre RMS Norm	Relative	-	-	80	64	8192	-
Galactica	Causal decoder	120B	Pre Layer Norm	Learned	GeLU	×	96	80	10240	2048
LaMDA	Causal decoder	137B	-	Relative	GeGLU	-	64	128	8192	-
Jurassic-1	Causal decoder	178B	Pre Layer Norm	Learned	GeLU	✓	76	96	13824	2048
LLaMA	Causal decoder	65B	Pre RMS Norm	RoPE	SwiGLU	✓	80	64	8192	2048
GLM-130B	Prefix decoder	130B	Post Deep Norm	RoPE	GeGLU	✓	70	96	12288	2048
T5	Encoder-decoder	11B	Pre RMS Norm	Relative	ReLU	×	24	128	1024	512

Tabelle 10 : Struktur von großen Modellen[46]

4.1.2 Vortrainingskorpus

Wie im Kapitel 3 erwähnt, erfordern große Sprachmodelle aufgrund ihrer zunehmenden Größe größere Trainingskorpora. Die folgende Tabelle zeigt die gängigen Korpora, die derzeit für das Training großer Modelle verwendet werden, sowie den Anteil der verschiedenen Arten von Textdaten in diesen Modellen.

Corpora	Size	Source	Latest Update Time
BookCorpus	5GB	Books	Dec-2015
Gutenberg	-	Books	Dec-2021
C4	800GB	CommonCrawl	Apr-2019
CC-Stories-R	31GB	CommonCrawl	Sep-2019
CC-NEWS	78GB	CommonCrawl	Feb-2019
REALNEWs	120GB	CommonCrawl	Apr-2019
OpenWebText	38GB	Reddit links	Mar-2023
Pushift.io	-	Reddit links	Mar-2023
Wikipedia	-	Wikipedia	Mar-2023
BigQuery	-	Codes	Mar-2023
the Pile	800GB	Other	Dec-2020
ROOTS	1.6TB	Other	Jun-2022

Tabelle 11 : Übliche Korpora[46]

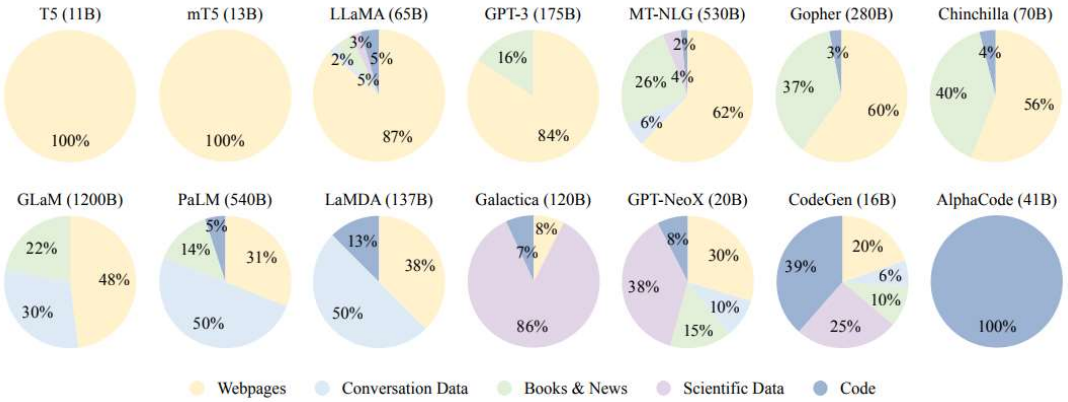


Abb.33 Verhältnisse unterschiedlicher Korpora bei Vortraining [46]

4.2 Feinabstimmung großer Sprachmodelle

Der Prozess der Feinabstimmung großer Sprachmodelle kann als Hinzufügen spezifischer Aufgabenkenntnisse auf das bereits erlernte allgemeine Sprachwissen des Modells betrachtet werden. Dabei werden die Parameter des Modells anhand von Daten spezifischer Aufgaben feinjustiert. Dieser Prozess kann auf Basis des vortrainierten Modells erfolgen, um eine bessere Anpassung an neue Aufgaben zu erreichen.

Jedoch gibt es auch Herausforderungen bei der Feinabstimmung großer Sprachmodelle.

Erstens erfordert die Feinabstimmung von großen Modellen erhebliche Rechenressourcen und Zeit. Zweitens kann die Feinabstimmung zu Overfitting führen, insbesondere wenn die Trainingsdaten für eine bestimmte Aufgabe relativ gering sind.

Um diese Probleme zu lösen, haben Forscher eine Reihe von Techniken vorgeschlagen. Zum Beispiel können mit den Techniken des Prompt Tuning und der Parameter-efficient Feinabstimmung vorgenommen werden, um den Ressourcenverbrauch zu reduzieren, während die meisten Modellparameter unverändert bleiben. Gleichzeitig können diese Methoden auch dazu beitragen, Überanpassungsprobleme effektiv zu mildern, da sie nur einen Teil der Modellparameter anpassen.

4.2.1 Prompt Tuning

Die grundlegende Idee von Prompt Tuning besteht darin, eine spezifische Anleitung für eine Ziel-Aufgabe zu erstellen, die dem Modell hilft, die Aufgabe besser zu verstehen und zu bearbeiten. Zum Beispiel können wir, wenn wir das Modell für eine Textklassifizierungsaufgabe trainieren möchten, eine Anleitung wie "Die Kategorie dieses Textes ist:" verwenden. Dann verwenden wir Trainingsdaten, um die Modellparameter anhand dieser Anleitung zu optimieren.

Die Hauptherausforderung bei Prompt Tuning besteht darin, effektive Prompts zu entwerfen, um das Modell besser auf die Ziel-Aufgabe auszurichten. Es ist auch notwendig, zu erforschen, wie das Modell trainiert werden kann, um die besten Prompt-Parameter zu erhalten. Liu et al.[52] geben in ihrer umfassenden Übersichtsarbeit zu Prompt-basiertem Lernen einen Klassifikationsgraphen für die Methoden des Promptings an.

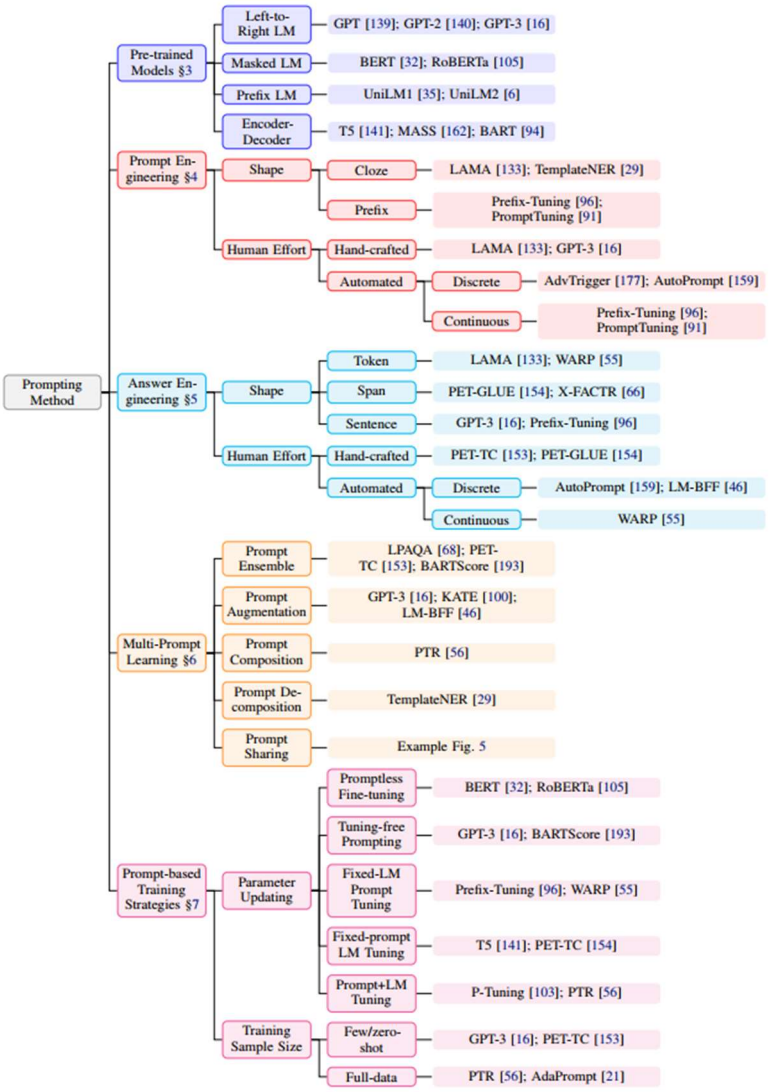


Abb.34 Kategorien von Prompt-Tuning [52]

4.2.2 Parameter-efficient Fine-tuning

Parameter-efficient Fine-tuning (parameter-effiziente Feinabstimmung) ist eine weitere Technik zur Feinabstimmung von Sprachmodellen. Der Hauptgedanke dieser Methode ist, dass nicht alle Modellparameter feinabgestimmt werden müssen, sondern nur eine Teilmenge der Schlüsselparameter optimiert werden soll.

Der Hauptvorteil dieser Methode besteht darin, dass sie erhebliche Ressourceneinsparungen ermöglicht. Im Vergleich zur Feinabstimmung aller Modellparameter ist es effizienter, nur eine kleine Teilmenge der Parameter zu optimieren.

In der [53] von He auf der ICLR 2022 wurde die typische PEFT-Methode untersucht. Das Ziel war es, PEFT in einem Framework zu vereinheitlichen, die genauen Gründe für ihre Wirksamkeit zu identifizieren und Verbesserungen vorzunehmen.

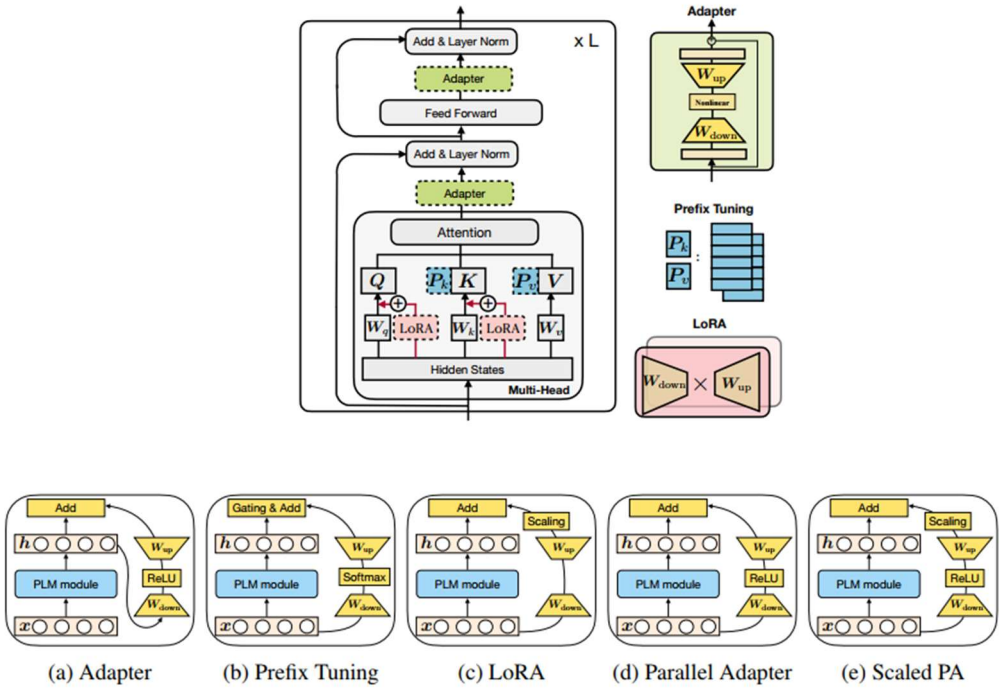


Abb.35 Kategorien von Parameter-efficient Fine-tuning [53]

4.3 Zusammenfassung und Ausblick auf die Entwicklung des NLP im Zeitalter großer Modelle

Seit der Einführung des Transformer-Modells vor sechs Jahren hat das NLP-Feld eine beispiellose Entwicklung durchlaufen. Der gesamte Prozess lässt sich in zwei Phasen einteilen:

Phase 1: Von Deep Learning bis hin zu zweistufigen Vortrainierten Modellen

Diese Phase umfasst den Zeitraum von der Einführung des Deep Learning in den Bereich der natürlichen Sprachverarbeitung (ca. 2013) bis zur Veröffentlichung von GPT-3 (Mai 2020).

Bevor BERT und GPT auftauchten, dominierten Deep Learning-Modelle das NLP-Feld. Diese Modelle basierten hauptsächlich auf verbesserten LSTM-Modellen und einigen verbesserten CNN-Modellen als Merkmalsextraktoren sowie auf der Kombination von Attention als typischem Technologie-Framework für verschiedene konkrete Aufgaben.

Obwohl die oben genannten Kerntechnologien ständig optimiert und verbessert wurden, war ihr Vorteil gegenüber nicht-deep-learning-basierten Methoden in Bezug auf die Lösung spezifischer Aufgaben nicht besonders deutlich. Dies kann auf zwei Hauptgründe zurückgeführt werden:

Begrenzte Datenmenge, die für spezifische Aufgaben zur Verfügung steht und die nicht mit dem zunehmenden Modellumfang mithalten kann.

Begrenzte Ausdrucksfähigkeit der LSTM/CNN-Merkmalsextraktoren, selbst bei ausreichender Datenmenge können sie das Wissen nicht effektiv aufnehmen und nutzen.

Die Einführung von BERT und GPT markierte einen Durchbruch im NLP-Bereich, sowohl aus wissenschaftlicher als auch aus industrieller Sicht. Die technischen Ansätze und Frameworks in verschiedenen Subbereichen der NLP begannen sich anzunähern, und die Forschungsrichtungen konvergierten weitgehend auf zwei Hauptarten von Techniken: **Natural Language Understanding** (NLU)-Aufgaben und **Natural Language Generation** (NLG)-Aufgaben.

NLU-Aufgaben umfassen Textklassifizierung, Satzbeziehungsbestimmung, Sentiment-Analyse usw., die im Wesentlichen Klassifikationsaufgaben sind. **NLG**-Aufgaben umfassen Chatbots, maschinelle Übersetzung, Textzusammenfassung, Frage-Antwort-Systeme usw., bei denen das Modell auf der Grundlage eines Eingabetextes einen Ausgabetext generiert. Der Hauptunterschied liegt in der Form des Eingabe- und Ausgabetextes.

Gleichzeitig wechselten die Merkmalsextraktoren in den verschiedenen NLP-Subbereichen allmählich von LSTM/CNN zum Transformer. Der Transformer hat nicht nur viele Bereiche der NLP vereinheitlicht, sondern wird auch allmählich andere Modelle wie CNN ersetzen, die in Bereichen wie der Bildverarbeitung weit verbreitet sind. Dieser Trend begann Ende 2020 mit dem Vision Transformer (ViT) und entwickelt sich weiterhin dynamisch.

In dieser Zeit wechselten die meisten NLP-Subbereiche zu einem zweistufigen Modell: Vortraining des Modells + Feinabstimmung.

Phase 2: Vom PLMs hin zur künstlichen Allgemeinen Intelligenz (AGI)

Diese Phase erstreckt sich von der Einführung von GPT3.0 (ungefähr Juni 2020) bis heute, und wir befinden uns immer noch in diesem Übergangsprozess.

Seit GPT-2 verfügen die Modelle der GPT-Serie über die Fähigkeit zum Zero-Shot/Few-Shot Prompting. Obwohl diese Fähigkeit damals noch nicht optimal war, wurde deutlich, dass es ideal wäre, Aufgaben auf diese Weise zu erledigen.

Ein ideales LLM-Modell, das dies erreichen kann, sollte zunächst über eine starke autonome Lernfähigkeit verfügen. Nehmen wir an, wir geben ihm alle verfügbaren Text- und Bilddaten auf der Welt. Es sollte in der Lage sein, automatisch daraus zu lernen und das gesamte Wissen zu speichern, um dieses Wissen flexibel zur Lösung realer Probleme anzuwenden. Da die Datenmenge massiv ist und eine große Anzahl von Modellparametern erfordert, ist ein solches Modell zwangsläufig riesig, und nur wenige Organisationen haben die Möglichkeit, diese Modellparameter zu erstellen oder anzupassen. Die Aufgabensteller, die diese Modelle verwenden möchten, können jedoch zahlreich sein, von kleinen und mittleren Unternehmen bis hin zu Einzelpersonen. Selbst wenn das Modell Open Source ist, können Benutzer es nicht bereitstellen, geschweige denn die Modellparameter mit Fine-Tuning ändern. Daher hoffen die Menschen, eine Möglichkeit zu haben, Aufgaben zu erledigen, ohne die

Modellparameter zu ändern, indem sie die Prompting-Methode anstelle des Feinabstimmung-Modus verwenden.

Die Modellentwickler sollten LLM als öffentlichen Service anbieten, der als Service ausgeführt wird, wie zum Beispiel ChatGPT. Angesichts der Vielfalt der Benutzeranforderungen sollten die Modellentwickler danach streben, LLM in der Lage zu machen, so viele Aufgabentypen wie möglich zu bewältigen. Wenn dieser Wunsch wirklich umgesetzt wird, wird das große Sprachmodell zu einem Modell für Künstliche Allgemeine Intelligenz (AGI). Es ist der Benutzerbedarf, der die kontinuierliche Weiterentwicklung des großen Sprachmodells vorantreibt.

Ob es sich um Zero-Shot-Prompting, Few-Shot-Prompting oder die Chain-of-Thought (CoT)-Prompting-Technik[54] handelt, sie sind allesamt vorhandene Techniken der Mensch-Maschine-Schnittstelle. Genauer gesagt, wurde Zero-Shot-Prompting mit der Absicht entwickelt, eine ideale Mensch-Maschine-Schnittstelle zu schaffen, bei der das LLM Aufgaben auf eine Weise ausführt, die dem menschlichen Verhalten entspricht. Es stellte sich jedoch heraus, dass das LLM diese Art der Anweisung oft nicht gut versteht und die Ergebnisse nicht zufriedenstellend sind. Durch Forschung wurde festgestellt, dass es für eine bestimmte Aufgabe besser ist, dem LLM einige Beispiele zur Verfügung zu stellen, die die Aufgabenbeschreibung repräsentieren. Aus diesem Grund konzentrieren sich die Forschungsanstrengungen auf die Verbesserung der Few-Shot-Prompting-Techniken.

Daher können wir sagen, dass Few-Shot-Prompting nur eine Übergangstechnik ist. Wenn eine Aufgabe auf eine natürlichere Weise beschrieben werden kann und das LLM diese Beschreibung verstehen kann, werden die Menschen zweifellos die Übergangstechniken aufgeben, da sie nicht den menschlichen Nutzungsgewohnheiten entsprechen.

Mit dem Aufkommen von ChatGPT beginnen die Menschen die Pforten zur AGI zu öffnen. Als derzeit fortschrittlichste Methode, die dem idealen LLM am nächsten kommt, zeigt ChatGPT eine beeindruckende Leistung und ein gutes Verständnis der Benutzeranforderungen.

Die beeindruckende Leistung des ChatGPT durch die Kombination von umfangreichen Datensätzen und einer großen Modellstruktur überrascht nicht. Aber das Verständnis der Benutzeranforderungen durch ChatGPT basiert auf dem neuen Wissen, das OpenAI in das GPT 3.5-Modell eingeführt hat. Dieses Wissen ist kein Weltwissen, sondern menschliches Präferenzwissen, einschließlich der gängigen Ausdrücke für Aufgabenstellungen und der Präferenzen der Menschen hinsichtlich der Qualität der Antworten. Auf diese Weise ermöglicht ChatGPT dem LLM, menschliche Befehle zu verstehen und verbessert damit die Benutzerfreundlichkeit und das Benutzererlebnis des LLM.[55] Dies wird zukünftige LLM-Modelle dazu inspirieren, weitere Verbesserungen in der Mensch-Maschine-Schnittstelle vorzunehmen und das LLM noch "gehorchsam" zu machen.

Kapitel 5 Zusammenfassung und Ausblick

Dieser Artikel hat aus der Perspektive der Modellanwender umfassend die Struktur des Transformer-Modells und seine Bedeutung in der Entwicklung des NLP vorgestellt. Insbesondere wurde detailliert auf die Prinzipien der Self-Attention-Mechanismen und deren Anwendung in der natürlichen Sprachverarbeitung eingegangen.

Anschließend wurden einige repräsentative Pretrained-Modelle wie BERT, GPT und T5 diskutiert. Ihre einzigartigen Designs und ihre Wirksamkeit haben eine wichtige Bedeutung für das Verständnis der Transformer-Struktur und den Wert von Pretrained-Modellen.

Schließlich wurde die Diskussion auf Large Language Models (LLM) gelenkt. Die Einführung von GPT-3 als Vertreter von LLM hat zu signifikanten Verbesserungen in Bezug auf Größe und Leistung geführt, was einen tiefgreifenden Einfluss auf das Gebiet der natürlichen Sprachverarbeitung hatte. Diese Informationen sind hilfreich, um moderne NLP-Techniken zu verstehen und anzuwenden.

In den letzten Jahren haben große Sprachmodelle, die auf der Transformer-Architektur basieren, in verschiedenen Aufgaben eine beeindruckende Leistung gezeigt. Dennoch gibt es noch viele Forschungsbereiche, die weiter erkundet werden müssen, wenn es um diese leistungsstarken Modelle geht.

Erforschung der Skalierungsgrenzen von LLM-Modellen

Obwohl große Sprachmodelle bereits große Erfolge erzielt haben, ist die Frage nach den Skalierungsgrenzen dieser Modelle immer noch offen. Zukünftige Forschung sollte untersuchen, wie sich die Leistung der Modelle mit zunehmender Größe verändert und nach der optimalen Modellgröße suchen. Darüber hinaus müssen effizientere Trainingstechniken und -algorithmen entwickelt werden, um größere Modelle zu trainieren und die Rechenressourcenbeschränkungen zu überwinden.

Stärkung der komplexen Inferenzfähigkeit von LLMs

Obwohl LLMs in vielen Aufgaben eine überlegene Leistung zeigen, gibt es immer noch Verbesserungsmöglichkeiten bei Aufgaben, die komplexe Inferenz erfordern. Zukünftige Forschung sollte sich darauf konzentrieren, Techniken und Methoden zu erforschen, wie externe Wissensquellen eingebunden werden können oder wie effektivere Trainingsmethoden entwickelt werden können, um die Leistung von LLMs in komplexen Inferenzaufgaben zu verbessern.

Erweiterung der Anwendung von LLMs auf andere Forschungsbereiche

Obwohl LLMs hauptsächlich im Bereich der natürlichen Sprachverarbeitung (NLP) eingesetzt werden, haben sie das Potenzial, weit über diesen Bereich hinaus angewendet zu werden. Zukünftige Forschung sollte erkunden, wie LLMs auf andere Bereiche angewendet werden können, um ihr Anwendungsspektrum zu erweitern.

Benutzerfreundlichere Mensch-Maschine-Schnittstellen

Um mehr Benutzern die bequeme Nutzung von LLMs zu ermöglichen, müssen benutzerfreundlichere Schnittstellen entworfen werden. Dies umfasst die Optimierung von Eingabe- und Ausgabeformaten, um Benutzern die bequeme Bereitstellung von Anweisungen und den Erhalt von Ergebnissen zu ermöglichen. Darüber hinaus sollten bessere Fehlerfeedback-Mechanismen entwickelt werden, um Benutzern das einfache Korrigieren von Modellfehlern zu ermöglichen.

Hochwertige Datenverarbeitung

Um bessere LLMs zu trainieren, werden hochwertige Trainingsdaten in großer Menge benötigt. Daher sollte zukünftige Forschung sich eingehender mit der hochwertigen Datenverarbeitung befassen, einschließlich Datensammlung, Datenbereinigung, Datenannotation und anderen Aspekten.

Zusammenfassend lässt sich sagen, dass obwohl große Sprachmodelle bereits beachtliche Erfolge erzielt haben, gibt es immer noch viele Bereiche für zukünftige Forschung. Der Tag, an dem die umfassende Anwendung von Künstlicher Allgemeiner Intelligenz (AGI) erreicht wird, rückt jedoch näher.

Literaturverzeichnis

[1] Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, *77*(2), 257-286.

[2] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, *26*.

[3] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.

[4] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

[5] Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training.

[6] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... & Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

[7] Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., ... & Sutskever, I. (2021, July). Learning transferable visual models from natural language supervision. In *International conference on machine learning* (pp. 8748-8763). PMLR.

[8] Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., ... & Sutskever, I. (2021, July). Zero-shot text-to-image generation. In *International Conference on Machine Learning* (pp. 8821-8831). PMLR.

[9] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, *323*(6088), 533-536.

[10] Elman, J. L. (1990). Finding structure in time. *Cognitive science*, *14*(2), 179-211.

[11] http://hemingwang.blogspot.com/2019/09/lstm.html

[12] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

[13] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

[14] https://zhuanlan.zhihu.com/p/26218969

[15] http://jalammar.github.io/illustrated-transformer/

[16] Pennington, J., Socher, R., & Manning, C. D. (2014, October). Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP) (pp. 1532-1543).

[17] Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2017). Enriching word vectors with subword information. Transactions of the association for computational linguistics, 5, 135-146.

[18] https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

[19] Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473.

[20] https://speech.ee.ntu.edu.tw/~hylee/ml/ml2021-course-data/self_v7.pdf

[21] He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep residual learning. Image Recognition, 7.

[22] "Common crawl." [Online] Available: https:// commoncrawl.org/

[23] "Bookcorpus" [Online] Available: https://huggingface.co/datasets/bookcorpus

[24] "Reddit Conversations" [Online] Available: https://paperswithcode.com/dataset/reddit-conversation-corpus

[25] "Wikipedia" [Online] Available: https://huggingface.co/datasets/wikipedia

[26] "PubMed" [Online] Available: https://huggingface.co/datasets/pubmed

[27] https://arxiv.org/

[28] "Github Code Dataset" [Online] Available: https://huggingface.co/datasets/codeparrot/github-code

[29] Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., & Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. Advances in neural information processing systems, 32.

[30] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., ... & Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. The Journal of Machine Learning Research, 21(1), 5485-5551.

[31] Clark, K., Luong, M. T., Le, Q. V., & Manning, C. D. (2020). Electra: Pre-training text encoders as discriminators rather than generators. arXiv preprint arXiv:2003.10555.

[32] Qiu, X., Sun, T., Xu, Y., Shao, Y., Dai, N., & Huang, X. (2020). Pre-trained models for natural language processing: A survey. Science China Technological Sciences, 63(10), 1872-1897.

[33] Yang, J., Jin, H., Tang, R., Han, X., Feng, Q., Jiang, H., ... & Hu, X. (2023). Harnessing the power of llms in practice: A survey on chatgpt and beyond. arXiv preprint arXiv:2304.13712.

[34] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., ... & Dean, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. arXiv preprint arXiv:1609.08144.

[35] CC_News https://huggingface.co/datasets/cc_news

[36] Openwebtext https://huggingface.co/datasets/openwebtext

[37] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. OpenAI blog, 1(8), 9.

[38] CC-Stories https://paperswithcode.com/dataset/cc-stories

[39] https://gluebenchmark.com/

[40] Sarzynska-Wawer, J., Wawer, A., Pawlak, A., Szymanowska, J., Stefaniak, I., Jarkiewicz, M., & Okruszek, L. (2021). Detecting formal thought disorder by deep contextualized word representations. Psychiatry Research, 304, 114135.

[41] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., ... & Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692.

[42] Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., & Soricut, R. (2019). Albert: A lite bert for self-supervised learning of language representations. arXiv preprint arXiv:1909.11942.

[43] He, P., Liu, X., Gao, J., & Chen, W. (2020). Deberta: Decoding-enhanced bert with disentangled attention. arXiv preprint arXiv:2006.03654.

[44] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. Advances in neural information processing systems, 33, 1877-1901.

[45] Child, R., Gray, S., Radford, A., & Sutskever, I. (2019). Generating long sequences with sparse transformers. arXiv preprint arXiv:1904.10509.

[46] Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., ... & Wen, J. R. (2023). A survey of large language models. arXiv preprint arXiv:2303.18223.

[47] Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., ... & Fiedel, N. (2022). Palm: Scaling language modeling with pathways. arXiv preprint arXiv:2204.02311.

[48] Rae, J. W., Borgeaud, S., Cai, T., Millican, K., Hoffmann, J., Song, F., ... & Irving, G. (2021). Scaling language models: Methods, analysis & insights from training gopher. arXiv preprint arXiv:2112.11446.

[49] Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., ... & Sifre, L. (2022). Training compute-optimal large language models. arXiv preprint arXiv:2203.15556.

[50] Thoppilan, R., De Freitas, D., Hall, J., Shazeer, N., Kulshreshtha, A., Cheng, H. T., ... & Le, Q. (2022). Lamda: Language models for dialog applications. arXiv preprint arXiv:2201.08239.

[51] Artetxe, M., Du, J., Goyal, N., Zettlemoyer, L., & Stoyanov, V. (2022). On the Role of Bidirectionality in Language Model Pre-Training. arXiv preprint arXiv:2205.11726.

[52] Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., & Neubig, G. (2023). Pre-train, prompt,

and predict: A systematic survey of prompting methods in natural language processing. ACM Computing Surveys, 55(9), 1-35.

[53]He, J., Zhou, C., Ma, X., Berg-Kirkpatrick, T., & Neubig, G. (2021). Towards a unified view of parameter-efficient transfer learning. arXiv preprint arXiv:2110.04366.

[54] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Chi, E., Le, Q., & Zhou, D. (2022). Chain of thought prompting elicits reasoning in large language models. arXiv preprint arXiv:2201.11903.

[55] Ye, J., Chen, X., Xu, N., Zu, C., Shao, Z., Liu, S., ... & Huang, X. (2023). A comprehensive capability analysis of gpt-3 and gpt-3.5 series models. arXiv preprint arXiv:2303.10420.