technische universität
dortmund

Masterarbeit

Erkennung von Ähnlichkeiten zwischen mathematischen Ausdrücken mittels Bidirectional Encoder Representations from Transformers

Stefan Todorinski
März 2021

Gutachter:
Prof. Dr. Katharina Morik
Lukas Pfahler

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl für Künstliche Intelligenz
https://www-ai.cs.tu-dortmund.de

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Das hohe Tempo der Wissenschaft ist an die vielen monatlich veröffentlichen Artikel zu erkennen. Allein auf arXiv.org wurden mehr als 10.000 Artikel pro Monat im Jahr 2020 publiziert [1].

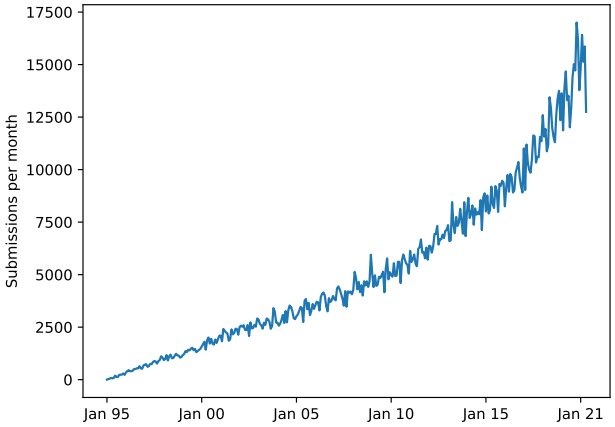

Abbildung 1.1: Anzahl publizierter Artikel pro Monat auf https://arxvix.org

In manche Felder wie dem maschinellen Lernen ist schwierig und oft sogar unmöglich die verwandte Literatur Aufmerksamkeit zu schenken, sie vollständig zu verstehen und ihre Ergebnisse praktisch zu reproduzieren. Die Unterstützung dabei ist die Nutzung von Suchmaschinen wie Google. Ihre Funktionalität ist aber vor allem an die Übereinstimmungen zwischen natürlichsprachlicher Eingabe und Textkorpus bedingt. Für Suchanfragen im Bereich der Mathematik ist die mathematische Notation eine bessere Datenrepräsentation, weil sie die visuelle Information und die Beziehungen zwischen Operatoren, Funktionen und Variablen beinhaltet. Dementsprechend besteht der Bedarf nicht nur in und mit natürlichsprachlichen Eingaben suchen zu können, sondern auch anhand von mathematischer Notation.

[1] https://arxiv.org/stats/monthly_submission

Im NLP-Bereich werden Transformer-Modelle und vor allem BERT (Bidirectional Encoder Representations from Transformers) als Transfer Lerning Basis benutzt, weil sie ein grundlegendes Verständnis für eine natürliche Sprache aufbauen und sich damit anschließend auf konkrete Aufgaben spezialisieren lassen können. Dieser Ansatz ist auf mathematische Formeln auch anwendbar. Durch eine Pre-Training-Phase, wo das Vorhersagen von maskierten Symbolen unter anderem Operatoren, Funktionen oder Variablen und das Einschätzen der Kohärenz von Formeln erlernt wird, wird ein grundlegendes Verständnis für die Mathematik aufgebaut. Das ist vom Vorteil, wenn sich das Modell auf weitere mathematische Aufgaben wie das Suchen nach und mit mathematischen Formeln, die Korrektur von unvollständigen oder inkorrekten Ausdrücken, das diskriminative oder generative Lösen von Gleichungen, die Gruppierung mathematischer Literatur und die allgemeine Klassifizierung von Formeln erweitern lässt. Im Allgemein kann jede Verarbeitung mathematischer Formeln von einem grundlegenden Verständnis für die Mathematik profitieren.

1.2 Aufbau

In Rahmen dieser Masterarbeit wird das BERT-Modell von Google inklusive seiner Architektur, Schichten, Optimierungsverfahren und Datenvorbereitung implementiert und mit dem arXiv-Datensatz trainiert. Dabei werden drei Hypothesen getestet. Die erste untersucht, ob die auf die Architektur bezogenen Parameter wie Schichtenanzahl und versteckte Größe positiv mit der Modellgenauigkeit korrelieren und ob eine obere Schranke für die Modellgenauigkeit besteht, die durch eine Erhöhung der Parameteranzahl nicht zu überwinden ist. Die zweite Hypothese testet, ob die Datenrepräsentation von mathematischen Formeln entscheidend für die Modellgenauigkeit ist. Hier wird geprüft, ob die Knotenbeziehungen stärker diese beeinflussen als die strikte Ordnung. Die Größe des Vokabulars und die Sequenzlänge haben direkten Einfluss auf die Trainingsdauer und werden so miniert, dass die Modellgenauigkeit möglichst hoch bleibt. Die dritte Hypothese untersucht, ob die Mathematik ein Few-Shot-Learner ist indem pre-trained und nicht pre-trained Modelle auf das diskriminative und generative Ableiten von Polynomen fine-tuned werden. Dabei werden nur wenig Daten benutzt und für wenige Trainingsiterationen optimiert. Transformer Modelle haben eine große Anzahl an Parametern, großen Speicherbedarf und benötigen viel Berechnungsaufwand. Ziel ist schnell und mit wenig Aufwand ein fine-tuned Modell erschaffen. Beim Pre-Training besteht der Bedarf das Modell verteilt über mehrere GPUs zu trainieren und den Einfluss des verteilten Trainings auf die Trainingsdauer zu untersuchen.

Kapitel 2

Verwandte Arbeiten

Es bestehen verschiedene Ansätze für die Verarbeitung mathematischer Formeln und einige davon ähneln stark dem Ansatz dieser Masterarbeit. In diesem Kapitel wird ein Einblick in einige dieser Ansätze gegeben.

2.1 Computer Vision Ansätze

Ein erster Ansatz zum Suchen nach Ähnlichkeiten zwischen mathematischen Formeln schlagen Pfahler et al. (2019) vor. Sie benutzen Convolutional Neural Networks, um aus der visuellen mathematischen Repräsentation Merkmale zu extrahieren, die in einem Vektorraum so kodierbar sind, dass mit einem Ähnlichkeitsindikator die Ähnlichkeit von Formeln messbar ist. Der Vorteil des Ansatzes ist, dass der benutzte Ähnlichkeitsmaß keine expliziten Annotationen benötigt. Als Labels werden bestehende Informationen aus den Datensätzen wie LaTeX-Codes, Abstract-Keywords und Dokumentenkontext verwendet. Somit zählt der Ansatz zu der Kategorie des semi-überwachten Lernens. Die Autoren bilden drei Modelle - für jedes Label eins. Das erste benutzt Ausdrücke aus den LaTeX-Codes als Labels, das zweite Schlüsselwörter aus dem Abstract und das letzte die Zugehörigkeit der Formeln - ob zwei Formeln aus demselben Artikel stammen. Der Ansatz ähnelt dem von Mikolov et al. (2013), wo anhand eines bestehenden Zusammenhangs ein nicht bekannter erlernt wird, der aber bestimmte Ähnlichkeiten zu dem bestehenden Zusammenhang aufweist.

2.2 Graph-basierte Ansätze

Convolutional Neural Network optimieren im Verlauf des Trainings ihre Filter für die Erkennung lokaler Muster. Oft sind aber globale Muster entscheidender. Zum Erlernen globaler Muster werden die Graph Convolutional Networks benutzt. Sie eignen sich gut dazu Beziehungen von Graphknoten zu erlernen. Entsprechend sind diese

Networks auch auf mathematische Formeln anwendbar und werden von Pfahler and Morik (2020) untersucht. Die Grundidee ihr Ansatz ist, dass jede mathematische Formel auch als Graphen präsentierbar ist. Die Autoren transformieren die XML-Struktur von mathematischen Formeln in einer Adjazenzmatrix für die Beziehungen und einer Merkmalsmatrix für die Repräsentation der Knoten. Dabei wird jeder Knoten anhand einer Einbettungsmatrix und eines Vokabulars kodiert wird. Ähnlich dem Ansatz von Pfahler et al. (2019) werden hier keine expliziten Annotationen, sondern die Zugehörigkeiten zum selben Artikel verwendet, was die Arbeit der Kategorie des semi-überwachten Lernens ordnet. In dem beschrieben Ansatz wird eine Maskierung von Tokens durchgeführt und diese in die Minimierungsfunktion integriert. Diese Erweiterung dient dem Erlernen von kontextuellen Ähnlichkeiten und bringt dem Modell bei Unvollständigkeit von Formeln zu korrigieren, was die Generalisierung verbessert. Die Tokens innerhalb einer Formel haben Abhängigkeiten zwischen sich selbst und diese sollen erlernt werden. Die Erweiterung ähnelt dem Ansatz von Devlin et al. (2018).

2.3 Transformer-basierte Ansätze

Für die Verarbeitung mathematischer Formeln haben Lample and Charton (2019) ein Ansatz zum Lösen von differential Gleichungen vorgestellt, der sehr stark dem Ansatz dieser Masterarbeit ähnelt. Sie bilden auch die Baumstruktur mathematischer Formeln auf Sequenzen ab, konstruieren Paare von Sequenzen - dem mathematischen Ausdruck und seiner Lösung - und lassen dann ein Transformer-Modell die Beziehungen zwischen den Paaren erlernen. So erlernt das Modell das Integrieren und das erste und zweite Ordnung Ableiten. Für die Repräsentation mathematischer Formeln verwenden die Autoren Pre-Order Datenrepräsentation. Für jede der Aufgaben besteht der Datensatz aus 40 Millionen Sequenzen mit einer Eingabelänge von höchstens 508 Tokens und einer Ausgabelänge von höchstens 335 Tokens. Die Beziehungen zwischen den Ausdrücken und den Lösungen werden nicht diskriminativ, sondern generativ erlernt. So ist das resultierende Modell in der Lage direkt die korrekte Lösung zu generieren und schafft es in Rahmen der Test-Phase genauere Resultate zu liefern als Systeme wie Matlab und Maple.

Transformer-basierte Modelle werden von Kim et al. (2020) für Code Vervollständigung eingesetzt. Anhand des Python-Abstract-Syntatic-Tree Datensatzes treffen sie bestehende RNN-Ansätze über. Sie benutzen Transformer-Modell mit sechs Schichten, einer Sequenzlänge von 1000 und einer versteckten Größe von 300 mit dem Ziel das nächste Token eines Programms anhand aller vorherigen Tokens vorherzusagen. Das wird durch eine Wahrscheinlichkeitsverteilung über das Vokabular aller Tokens ausgegeben, welches aus 100.000 Tokens besteht. Das Interessanteste

aus sicher dieser Masterarbeit ist, dass sie drei verschiedene Ansätze zur Datenrepräsentation vergleichen. Der erste Ansatz - SeqTrans - kodiert das Eingabeprogramm als Sequenz ohne die Baumbeziehungen in irgendeine Form zu betrachten. Die weiteren zwei Ansätze - PathTrans und TravTrans - betten in die Datenrepräsentation die Baumstruktur ein. Der erste kodiert jedes Blatt und sein Pfad zum Wurzel. Der zweite führt eine Pre-Order-Iteration durch und anhand dieser speichert die Knoten. Die Autoren zeigen, dass in Rahmen der Evaluation der TravTrans-Ansatz die besten Resultate erreicht.

Kapitel 3

Hypothesen

In Rahmen dieses Kapitels werden die drei im Rahmen der Masterarbeit getesteten Hypothesen vorgestellt.

3.1 Parameterzahl

Google BERT und OpenAI GPT-3 treffen die Annahme, dass die Parameteranzahl des Modells zu den wichtigsten Faktoren für die Modellgenauigkeit zählt. Es besteht die Überzeugung, dass ein grundlegendes Verständnis für Sprachen nur so zu erreichen ist und entsprechend hat sich von 2019 bis 2021 die Parameteranzahl dieser Modelle 5.000 Mal erhöht.

Jahr	Modell	Parameteranzahl
2019	Google BERT	330M
2020	OpenAI GPT-3	175B
2021	Google Switch-C	1.5T

Abbildung 3.1: Entwicklung der Parameteranzahl von 2019 bis 2021

Diese Tendenz soll kritisch und nicht als der primäre Faktor für Modellgenauigkeit betrachtet wie Bender et al. (2021) vorschlagen. Die Vermutung in dieser Masterarbeit ist, dass ab einem bestimmten Punkt eine obere Schränke erreicht wird, die durch Parametervergrößerung nicht zu überwinden ist und so jede Modellvergrößerung überflüssig macht. Für kleine Modelle, die nicht genug Kapazität zum Lösen der Aufgabe haben, ist Vergrößerung die einzige Option, aber für große Modelle mit genug Kapazität nicht unbedingt. Dementsprechend beginnen wir in Rahmen dieser Masterarbeit mit einem genug großen Modell als Basiskonfiguration. Die erste Hypothese lautet:

> **I.** Für die Modellgenauigkeit besteht eine obere Schranke, die nur durch Erhöhung der Parameteranzahl nicht zu überwinden ist.

3.2 Datenrepräsentation

Wenn mathematische Formeln aus ihrer Baumstruktur in eine sequentielle Form überführt werden, müssen die Beziehungen zwischen Knoten und Kinderknoten, die Richtung des Lesens und vor allem die Operatoren, Funktionen und Variablen möglichst unverändert repräsentiert. Die Datenrepräsentation ist entscheidend für die Modellgenauigkeit. Es gibt aber sehr viele Möglichkeiten mathematische Formeln zu repräsentieren. Wenn diese gelesen werden, ist die richtige Reihenfolge der Operatoren, Funktionen und Variablen entscheidend dafür, was unter der Formel verstanden wird. Wenn Symbole in die Formeln vertauscht werden, dann können ungültige oder inkorrekte Ausdrücke entstehen. Dementsprechend wird die Annahme getroffen, dass die Reihenfolge zu den wichtigsten Merkmalen der Datenrepräsentation zählt. Es werden zwei Datenrepräsentationen getestet. Die erste ist stärker auf die Reihenfolge der Formeln und die zweite auf die Knotenbeziehungen fokussiert. Mit Knotenbeziehungen sind die Relationen zwischen den mathematischen Symbolen gemeint und diese sind sehr wichtig, denn wenn sie nicht repräsentiert sind, dann ist die mathematische Bedeutung unvollständig. Die zweite Hypothese lautet:

> **II.** Die Reihenfolge ist entscheidender für die Modellgenauigkeit als die Knotenbeziehungen.

3.3 Few-Shot-Learning

Sprachmodelle wie BERT und GPT-3 treffen die Annahme, dass die natürlichen Sprachen Few-Shot-Learner sind Brown et al. (2020). Das bedeutet, dass es eine Garantie für erfolgreiches Fine-Tuning besteht, auch wenn mit wenig Daten trainiert wird, solange das Pre-Training mit viel Daten und genug Parametern gemacht ist. Grund dafür soll sein, dass das Pre-Training ein grundlegende Verständnis für die Aufgabe erlernt. In Rahmen dieser Masterarbeit wird diese Behauptung untersucht, indem die Modelle einmal pre-trained und einmal nicht pre-trained auf eine diskriminative und eine generative Aufgabe fine-tuned werden. Dabei werden nur wenig Daten verwendet und wenig Trainingsaufwand getrieben. Die dritte Hypothese ist wie folgt formuliert:

> **III.** Mathematik ist Few-Shot-Learner.

Kapitel 4

Bidirectional Encoder Representations from Transformers

Das BERT-Modell - Bidirectional Encoder Representations from Transformers - erreicht zum Zeitpunkt seiner Veröffentlichung in eine breite Anzahl an NLP-Aufgaben die besten Resultate Devlin et al. (2018). Im Januar 2021 wird das Modell in Glubenchmark (2020) und SQuAD2 (2020) von den Top Modellen jeweils verwendet und seine Architektur ist in Modelle wie GPT-3 von Brown et al. (2020) zu sehen. Es wird nicht nur für Forschung, sondern auch in Produktion wie zum Beispiel in die Google Search eingesetzt. In Rahmen dieses Kapitels werden die Grundideen, die Architektur, die Schichten und die Optimierungsverfahren vom BERT vorgestellt.

4.1 Idee und Architektur

Für Aufgaben in Domäne mit Mängeln an annotierten Daten ist es schwierig oder sogar unmöglich Modelle mit einer guten Generalisierung und Genauigkeit zu erschaffen. Um das zu lösen wird Transfer Learning eingesetzt. Das Transfer Learning ist ein Ansatz zur Wiederverwendung von Modellen, der der algorithmischen Reduktion ähnelt. Er versucht eine Aufgabe $\mathcal{T}_s$ mithilfe der Lösung für eine andere Aufgabe $\mathcal{T}_r$ zu lösen Tan et al. (2018). Das besondere ist, dass häufig die Aufgabe $\mathcal{T}_r$ mit vielen annotierten Daten bereits erfolgreich gelöst ist und das resultierte Modell ein grundlegendes Verständnis für die Daten aufgebaut hat. Das kann vom Nutzen für die Aufgabe $\mathcal{T}_s$ sein, wenn eine gewisse Ähnlichkeit zwischen $\mathcal{T}_r$ und $\mathcal{T}_s$ besteht. Das Transfer Learning ist weitverbreitet und wird für Computer Vision und Spracherkennung Aufgaben erfolgreich eingesetzt. Zum Beispiel zur Detektion von Objekten in YOLO v3 von Redmon and Farhadi (2018), Semantic Segmentation mit Mask R-CNN

von He et al. (2017) oder Spracherkennung Baevski et al. (2020).

Das Training von BERT funktioniert mit Transfer-Learning und besteht aus einer Pre-Training- und einer Fine-Tuning-Phase. Die erste dient zum Erlernen von intrinsischen Beziehungen und löst gleichzeitig zwei Aufgaben - maskierte Tokens vorherzusagen und die Kohärenz von Sätzen einzuschätzen. In die Fine-Tuning-Phase kann die Klassifikationsschicht konkret an einer bestimmten NLP-Aufgabe angepasst und das Model mit neuen Daten trainiert werden. Die Vermutung ist, dass die Pre-Training-Phase zur Verbesserung der Resultaten der Fine-Tuning-Phase führt, weil es für viele NLP-Aufgaben vom Vorteil ist, wenn das Modell ein Verständnis für die Kohärenz der Sätze erlernt und die Fähigkeit besitzt fehlende Lücke in Sätze zu ergänzen. Davon können Aufgaben wie Übersetzen, Antworten auf Fragen, Klassifikation und Erlernen weiterer Satzbeziehungen profitieren, weil all diese ein grundlegendes Verständnis für die Sprache benötigen.

Ein großer Vorteil während der Datenvorbereitung für das Pre-Training von BERT ist, dass es Daten aus Wikipedia und verschiedenen Buchquellen verwendet. Diese Daten haben bereits eine Gliederung. Es ist zu erkennen, welche Sätze aufeinanderfolgend sind. So werden es keine expliziten Annotationen benötigt. Das ordnet das Pre-Training von NLP-BERT der Kategorie des semi-überwachten Lernens. Die wissenschaftlichen Artikel aus arXiv, die in Rahmen dieser Masterarbeit verwendet werden, sind bereits so gegliedert, dass aus der Gliederungsstruktur auch wichtige Eigenschaften erkennbar und extrahierbar sind. Es ist zu erkennen, welche mathematische Formeln aus demselben Artikel stammen. So werden keine expliziten Annotationen benötigt. Das ordnet MATH-BERT auch der Kategorie des semi-überwachten Lernens.

BERT verwendet als Basisarchitektur die Transformer-Architektur von Vaswani et al. (2017). Diese besteht aus einem Encoder und einem Decoder. Diese bestehen jeweils aus mehreren Encoder- bzw. Decoder-Schichten. Das spezielle bei BERT ist, dass es im Vergleich zum ursprünglichen Transformer nur aus Encoder-Schichten besteht. Ein genauer Einblick in die Form einer Encoder-Schicht ist in Abbildung 4.1 zu sehen. Jede Encoder-Schicht besteht aus fünf Arten von Schichten - Multi Head Attention, Dense, Dropout, Residual, Batch Normalization und GeLU. Die Eingaben sind ein Batch aus Sequenzen und ein Batch aus Masken. Anhand der Eingaben wird in Rahmen der Multi Head Attention (Schicht 1) die Relevanz jedes Paar zweier Tokens der Sequenz ermittelt. In Schichten 2 bis 6 wird eine lineare Projektion durchgeführt. In Schichten 6 und 7 wird eine lineare Projektion auf einen höheren Raum durchgeführt und mit GeLU aktiviert. In Schichten 8 bis 11 wird alles zurück auf den Eingaberaum projiziert.

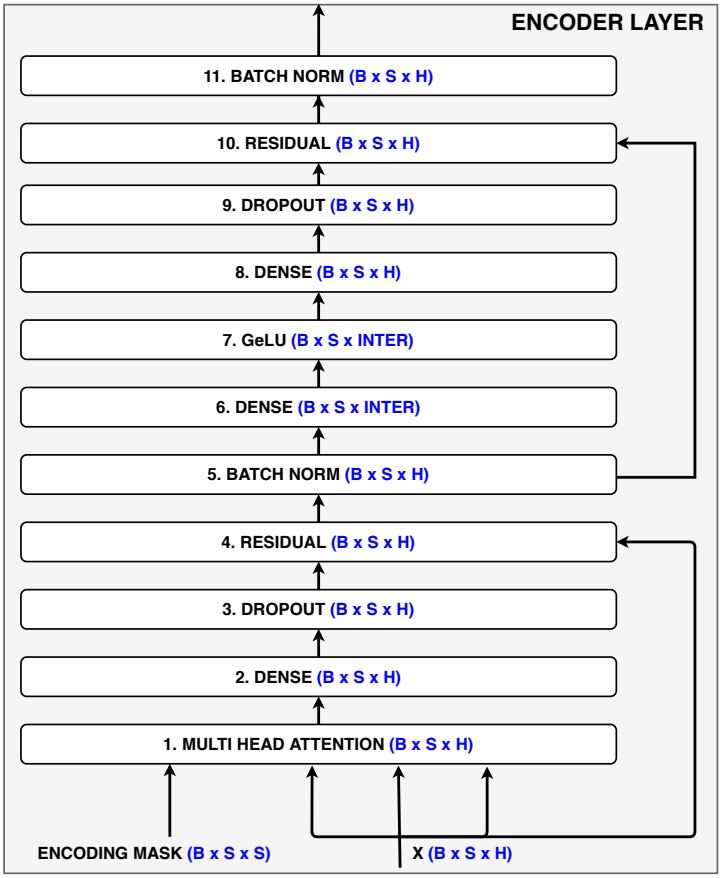

Abbildung 4.1: Encoder Layer nach BERT mit:
B = Batchgröße S = Sequenzlänge H = Versteckte Größe
N = Anzahl der Multi-Head Attention Heads D = Tiefe der Multi-Head Attention
INTER = Größe der Zwischenschicht

Die Anzahl an Encoder-Schichten zählt zu den Metaparametern von BERT. Diese ist in Abhängigkeit davon zu bestimmen, welche Ressourcen zur Verfügung stehen und was für eine Genauigkeit zu erzielen ist. In Devlin et al. (2018) werden zwei Konfigurationen vorgestellt - BERT-Base mit 12 Schichten (110M Parameter) und BERT-Large mit 24 Schichten (340M Parameter). Die Korrelation zwischen Schichtenanzahl und Genauigkeit ist positiv, aber klein. In manchen Aufgaben wie diese der Glubenchmark und SQuAD ist zu sehen, dass das BERT-Large weniger als 10% Anstieg an Genauigkeit hat, obwohl die Parameteranzahl um 300% steigt. Wie in Rogers et al. (2020) zu sehen ist, ist die Vielfalt an BERT-Konfiguration sehr umfangreich und gute Resultate auch mit kleineren Modellen zu erreichen sind. Die Suche nach dem perfekten Balance zwischen Größe und Genauigkeit ist eine der Herausforderungen des Modells. In GPT-3 wird die Annahme getroffen, dass die Parameteranzahl zu den wichtigsten Faktoren der Genauigkeit zählt, und werden Konstellationen mit bis zu 175 Milliarden Parametern erschaffen. Die Autoren sind auch davon überzeugt,

dass Few-Shot-Learning sehr effektiv sein kann, wenn die Große eines Transformer-Modells exorbitant wird Brown et al. (2020). Die Erhöhung der Parameterzahl wird von Fedus et al. (2021) auf mehr als 1 Trillion fortgesetzt.

Ein Gesamtblick in die BERT-Architektur ist in Abbildung 4.2 zu sehen. Außer des Encoders hat das Modell eine Einbettung- und eine Klassifikationsschicht. Beim Pre-Training hat BERT 3 Eingaben und 4 Labels. Die Eingaben sind die Eingabesequenz, die Eingabemaske und die Segmentation. Die Labels sind die maskierten Tokens, die maskierten Indizes, die maskierten Gewichten und die Sequenzbeziehung.

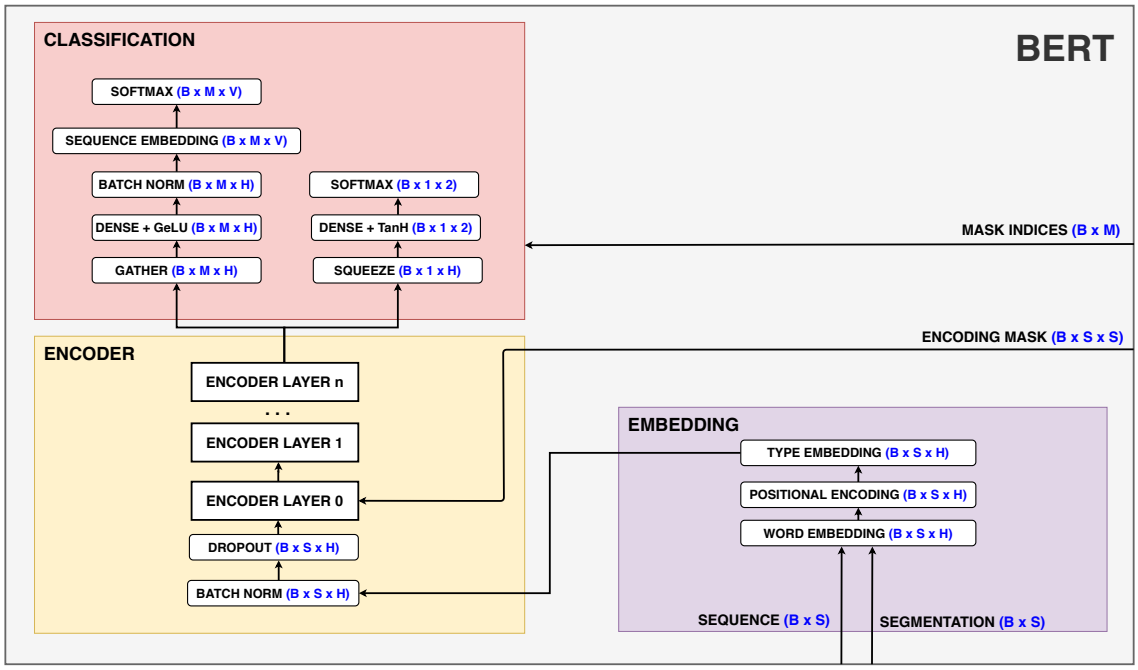

Abbildung 4.2: Architektur von BERT mit:
M = Anzahl der maskierten Tokens
B = Batchgröße V = Vokabulargröße S = Sequenzlänge H = Versteckte Größe

Beim Fine-Tuning werden die Eingaben und die Labels an die Aufgabe angepasst, aber versucht sie möglichst ähnlich zu halten oder sie so zu gestalten, dass sie dem Pre-Training-Format ähneln, denn andernfalls entsteht starke Abweichung von dem, was das Modell während der Pre-Training-Phase erlernt hat.

4.2 Worteinbettung

Die Worteinbettung im BERT wird benutzt, um jedes Token der Eingabesequenz auf einen Vektorraum so abzubilden, dass ähnliche Tokens einen möglichst kleinen Abstand in dem Vektorraum haben. So fokussiert sich das Modell stärker auf die Struktur der Tokens. Dabei sind ähnliche Tokens zum Beispiel diese, die in gleiche Konstellationen vorkommen. Zum Beispiel Die Tokens a und b, die beide als Nenner eines Quotienten in die folgenden zwei Abbildungen vorkommen, sollen aufgrund der ähnlichen Relationen als besonders ähnlich von der Worteinbettung interpretiert.

```
<mfrac>
  <mn>2</mn>
  <mi>a</mi>
</mfrac>
```

Abbildung 4.3: $2/a$

```
<mfrac>
  <mn>2</mn>
  <mi>b</mi>
</mfrac>
```

Abbildung 4.4: $2/b$

Die Worteinbettung hat die Signatur $\mathbb{N}^{B\times S} \to \mathbb{R}^{B\times S\times H}$ und funktioniert indem eine Gewichtsmatrix $W_{we} \in \mathbb{R}^{V\times H}$ aus einer Normalverteilung $\mathcal{N}(\mu = 0, \sigma^2 = 0.02)$ initialisiert wird. Mit der Gewichtsmatrix wird jedes Token $i \in I$ der Eingabesequenz $I \in \mathbb{N}^S$ auf den Vektor $W_{we,i} \in \mathbb{R}^{1\times H}$ abgebildet.

$$\text{Wordembedding}(I) = \text{gather}(W_{we}, I) \tag{4.1}$$

Die Gewichtsmatrix ist trainierbar und wird im Verlauf des Trainings optimiert. Die Ableitung der Worteinbettung ist gegeben durch den folgenden Ausdruck:

$$\frac{\partial \text{Wordembedding}(I)}{\partial W_{we}} = \frac{\partial \text{gather}(W_{we}, I)}{\partial W_{we}} = \text{gather}(\Delta, I) \tag{4.2}$$

Dabei bezeichnet Δ die Ableitung der folgenden Schicht. Die Auswahl der gleichen Indizes aus Δ und W_{we} bedeutet, dass nur die in der Sequenz vorkommenden Tokens bei der Berechnung des Gradienten berücksichtigt werden und somit die Gewichtsmatrix nur ihnen bezüglich optimiert wird. Aus diesem Grund ist es sinnvoll, dass das Vokabular das Verhältnis zwischen der Summe der akkumulierten Häufigkeiten und der Anzahl an Tokens maximiert, weil Tokens, die sehr selten vorkommen, auch sehr selten von dem Optimierer berücksichtigt werden. Das kann das Konvergieren verlangsamen oder sogar unmöglich machen. Dementsprechend ist es empfehlenswert, dass das Vokabular aus Tokens besteht, die eine hohe Häufigkeit aufweisen und somit repräsentativ sind. Der kleine Mittelwert und die kleine Standardabweichung der Initialisierung sichern die numerische Stabilität, so dass folgende Matrixmultiplikationen in Dense- oder Multi-Head-Attention-Schichten Vektoren mit kleinen Vektorlängen ausgeben. Dass bringt den Vorteil, dass die Gradienten und die Ausgaben schwieriger einen Zahlenübergang erreichen können.

4.3 Positional Encoding

Die Worteinbettung berücksichtigt die Struktur der Tokens, aber die Positionen der Tokens sind auch von Bedeutung, weil häufig Muster bestehen, die beim Vorhesagen von Tokens entscheidend sind. Zum Beispiel, wenn sich das Pronomen “Das” an der Stelle j im Satz befindet, dann ist zu erwarten, dass sich an der Stelle $j + 1$ ein Substantiv im Neutrum befindet. Die Beziehungen zwischen Tokens und ihren Positionen werden durch die Positional Encoding adressiert. Im Vergleich zu RNN und Convolutions haben Transformer Modelle keine explizite Information über die relative Position eines Tokens bzgl. der anderen Tokens der Sequenz. Dementsprechend muss diese explizit eingereicht werden. Das erfolgt dank der Positional Encoding.

Im ursprünglichen Transformer von Vaswani et al. (2017) wird als Positional Encoding eine auf trigonometrische Funktionen basierte Kodierung mit der folgenden Berechnungsvorschrift verwendet. $\mathbf{PE}(s, h)$ bezeichnet die h-te Kodierung des s-ten Tokens mit $s \in [0, S]$ und $h \in [0, H]$.

$$\mathbf{PE}(s, 2h) = \sin\left(\frac{s}{10000^{\frac{2h}{H}}}\right) \qquad \mathbf{PE}(s, 2h+1) = \cos\left(\frac{s}{10000^{\frac{2h}{H}}}\right) \tag{4.3}$$

Diese Kodierung hat den Vorteil, dass sie keine Annahme über die Ordnung der Tokens macht. Es wird nicht angenommen, dass die Positionen am Anfang oder am Ende eine besonders hohe oder kleine Korrelation mit dem Informationsinhalt haben. Ein genauer Einblick ist in die folgende Abbildung zu sehen. Es ist schwer Stellen in die Sequenz zu sehen (Zeilen im Bild), die ähnliche Kodierungen haben.

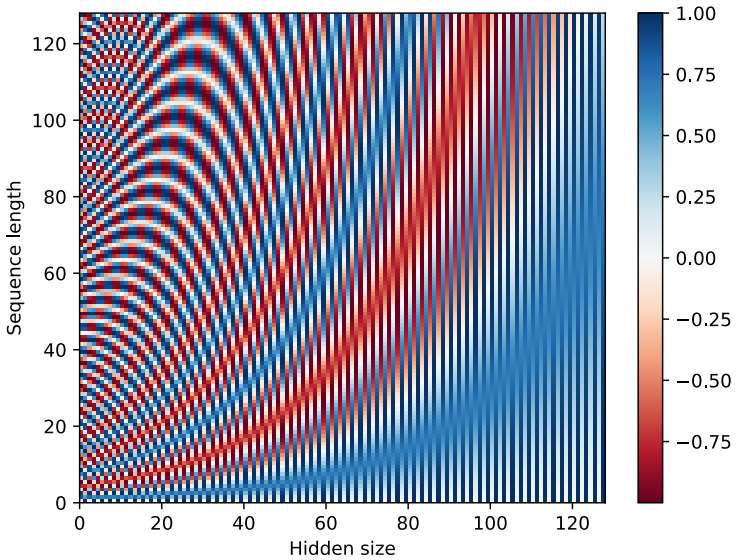

Abbildung 4.5: Positional Encoding Karte nach Vaswani et al. (2017) mit $S = 256$ und $H = 128$

Durch die Verwendung von sin und cos wird eine Periodizität und eine untere sowohl obere Schranke der Kodierungen erreicht. Das dient zur gleichmäßigen Streuung der Werte entlang der beiden Achsen. In Abbildung 4.5 sind hohe und kleine Kodierungswerte gleich über die Sequenzlänge gestreut. Die Schwankungen in die Amplituden (der Farbe) bringen mehr Eindeutigkeit der einzelnen Kodierungen und werden durch den exponentiellen Term sichergestellt. Die Einschränkung im Intervall $[-1, 1]$ dient zur Normalisierung der Werte. Jede Kodierung hat ab $H \geq 100$ eine Periodizität der Werte. Nichtsdestotrotz ist sie an jeder Sequenzposition anders. Nachteil dieser Kodierung ist, dass sie nicht trainierbar ist. So kann das Modell alle Abhängigkeiten, die durch die Berechnungsvorschrift nicht zu beschreiben sind, nicht berücksichtigen. Es werden keine neuen Abhängigkeiten erlernt.

Im BERT wird eine Positional Encoding verwendet, die sich stark von dieser im ursprünglichen Transformer unterscheidet. Im BERT werden die Sequenzpositionen mit einer Gewichtsmatrix W_{pe} kodiert, die aus der gleichen Normalverteilung wie die Worteinbettungsmatrix initialisiert wird - $\mathcal{N}(\mu = 0, \sigma = 0.02)$. Dabei wird jeder Eingabesequenz $I \in \mathbb{R}^{S \times H}$ mit Anzahl an relevanten Tokens $q \leq S$ die Teilmatrix $W_{pe,:q} \in \mathbb{R}^{q \times H}$ aufaddiert. Das bedeutet, dass $\mathsf{PE}(s, h)$ - die h-te Kodierung der s-ten Position in Sequenz ist und wie folgt definiert ist:

$$\mathsf{PE}(s,h) = W_{s,h} \quad s \leq q \qquad \mathsf{PE}(s,h) = 0 \quad s > q$$

$$\frac{\partial \mathsf{PE}(s,h)}{W_{s,h}} = \Delta_{s,h} \quad s \leq q \qquad \frac{\partial \mathsf{PE}(s,h)}{W_{s,h}} = 0 \quad s > q$$

Ziel dieser Kodierung ist die Tokens der Eingabesequenz stärker zu betonen und die Stellen außerhalb der Eingabesequenz zu dämpfen. Wie aus der Formel zu sehen ist, werden die Positionen $s \in S$, die selten in die Eingabesequenzen vorkommen, auch selten in die Gewichtsmatrix optimiert. Das ist dann der Fall, wenn die Länge einer Sequenz relativ zu allen anderen Sequenzenlänge länger ist. Deshalb ist empfehlenswert, dass die Anzahl an Sequenzlängen kürzer oder gleich der benutzen Sequenzlänge im Verhältnis zu der benutzen Sequenzlänge maximiert wird. Es gilt das folgende Maximierungsproblem:

$$arg\max_{\hat{S}} = \frac{\text{alle Sequenzen mit Länge } \leq \hat{S}}{\hat{S}}$$

Es wird die Sequenzlänge $\hat{S}$ gesucht, wo das Verhältnis zwischen allen Sequenzen, die durch $\hat{S}$ abgedeckt sind, und $\hat{S}$ maximiert wird. Mit Sequenzlänge ist die Anzahl der Tokens einer Sequenz gemeint. Der Vorteil der Positional Encoding im BERT ist, dass diese trainierbar ist. So können die Abhängigkeiten zwischen Positionen und Tokens erlernt werden, die durch eine konstante Positional Encoding nicht beschreibbar sind.

4.4 Multi Head Attention

Dense-Schichten mit Gewichtsmatrix $W \in \mathbb{R}^{H \times S}$, Aktivierungsfunktion $\theta : \mathbb{R}^{H \times B} \rightarrow \mathbb{R}^{H \times B}$, Eingabe $I \in \mathbb{R}^{S \times B}$, Bias $B : \mathbb{R}^{1 \times H}$ und Ausgabe $O \in \mathbb{R}^{H \times B}$ definiert als

$$O = \theta(W * I + B) \tag{4.4}$$

sind für sequentielle Daten nicht geeignet, denn es besteht durch die direkte Multiplikation mit den Gewichten W keinen Fokus auf die Reihenfolge von I, auch wenn entlang der S Dimension eine sequentielle Repräsentation kodiert ist.

Durch Umwandlung der Eingabe in sequentieller Form der Art $I \in \mathbb{R}^{Q \times S \times B}$ mit B die Batchgröße, S die Sequenzlänge und Q die Kodierung pro Token und Durchführung einer linken Reduktion entlang der S Dimension sind die Reccurent Neural Networks eine erste Möglichkeit das Modell stärker auf die Reihenfolge der Datenrepräsentation zu fokussieren. Die Ausgabe $h_{b,s} \in \mathbb{R}^{Q \times 1}$ bzw. Aktivierung $y_{b,s} \in \mathbb{R}^{Q \times 1}$ bzgl. einer Sequenz b und einer Position $s \in [0, S]$ in einem RNN-Netz sind wie folgt definiert:

$$\begin{aligned} h_{b,s} &= \sigma(U_h * I_{b,s} + W_h * h_{b,s-1} + b_h) \\ y_{b,s} &= \textsf{softmax}(W_y * h_{b,s} + b_y) \end{aligned}$$

Dabei sind Q die versteckte Größe des RNN, $b_h \in \mathbb{R}^{Q \times 1}$ bzw. $b_y \in \mathbb{R}^{Q \times 1}$ die Biase und $U_h \in \mathbb{R}^{Q \times H}$, $W_h \in \mathbb{R}^{Q \times Q}$ bzw. $W_y \in \mathbb{R}^{Q \times Q}$ die Gewichtsmatrizen. Die die Sequenzlänge S in einem RNN korreliert positiv mit dem Verschwinden der Gradienten, da σ eine Sigmoidfunktion ist. Ihre Ableitung ist $\sigma' \leq 0.25$. So ist die gesamte Ableitung bzgl. der Gewichte eine Faktorisierung mit σ' und kommt sehr nah an 0.

$$\begin{aligned} \frac{\partial h_{b,s}}{\partial W_h} &= \sigma'(U_h * I_{b,s} + W_h * h_{b,s-1} + b_h) \frac{\partial h_{b,s-1}}{\partial W_h} \\ \frac{\partial y_{b,s}}{\partial W_y} &= \left(\Delta_{b,s} \textsf{softmax}'(W_y * h_{b,s} + b_y)\right)^T * h_{b,s} \end{aligned}$$

RNN-Modelle werden für Spracherkennung und Computer-Vision in Deep Speech 2 von Amodei et al. (2015) und die NVidia DriveLab [1] erfolgreich benutzt, aber um die verschwindenden Gradienten zu lösen sind die LSTM (Long Short Term Memory) eingeführt. Sie führen in die Berechnug weitere Terme und die hyperbolische Aktivierungsfunktion ein, so dass die Wahrscheinlichkeit an verschwindenden Gradienten nicht so stark mit der Sequenzlänge korreliert. Allerdings benötigen die LSTMs mehr Speicher und Berechnungsaufwand, da mehr Gates berechnet werden. Das führt dazu, dass das Training langsamer und sensibler gegenüber der Initialisierung ist. Die LSTMs sind eine gute Basis für Zeitreihen und werden zum Beispiel erfolgreich

[1] https://blogs.nvidia.com/blog/2019/05/22/drive-labs-predicting-future-motion/

für Schrifterkennung in Tesseract 4.0 von Google verwendet [2]. Die RNNs und ihre Variationen folgen eine strikte Iteration - von links nach rechts oder umgekehrt. Der Informationsinhalt einer Sequenz kann aber gestreut sein und jedes Token kann mit jedem anderen in eine besondere Beziehung stehen.

Die Nachteile von RNNs sehen auch Vaswani et al. (2017) und schlagen den Einsatz von Multi Head Attention (MHA) vor. So können einige der Nachteile von RNNs gelöst werden. In MHA wird die Beziehung jeder zwei Tokens der Sequenz berücksichtigt. Außerdem wird mehr Stabilität beim Training eingeführt, indem keine Funktionen benutzt werden, die zu verschwindenden Gradienten führen. Weiterhin können Stellen der Sequenzen gedämpft werden, um den Fokus gezielt zu steuern. Die MHA lässt sich auch effizient berechnen lassen und kann von der modernen GPU- und TPU-Hardware profitieren. Die Architektur der Multi Head Attention ist in die folgende Abbildung zu sehen.

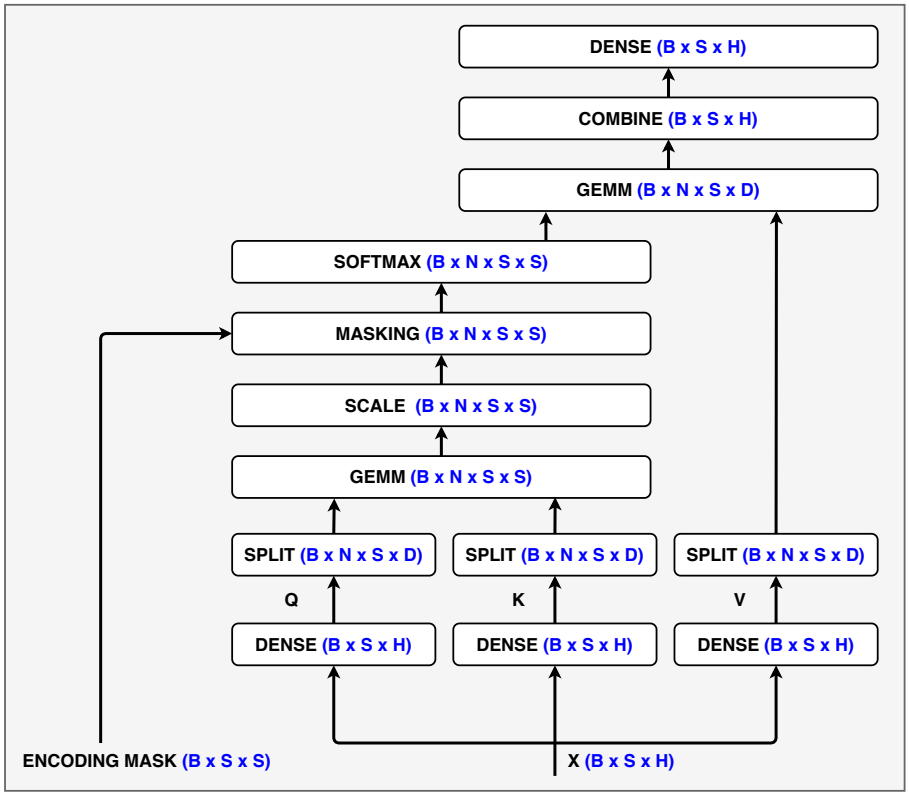

Abbildung 4.6: Multi Head Attention mit:
B = Batchgröße S = Sequenzlänge H = Versteckte Größe
N = Anzahl der Multi-Head Attention Heads D = Tiefe der Multi-Head Attention

[2] https://tesseract-ocr.github.io/tessdoc/4.0-with-LSTM

Die Multi Head Attention ist auf die Eigenschaft basiert, dass jede Matrixmultiplikation zwischen einer Matrix $M \in \mathbb{R}^{A \times B}$ und einer Matrix $V \in \mathbb{R}^{B \times C}$ mit Ausgabe $O \in \mathbb{R}^{A \times C}$ den größten Wert an die Stelle $M_i * M_j = O_{i,j}$ hat, wo der Winkel θ zwischen den Vektoren M_i und V_j minimal und die Längen $\|M_i\| = \sqrt{\sum_q M_{i_q}^2}$ und $\|V_j\| = \sqrt{\sum_q V_{j_q}^2}$ maximal sind, so dass das Skalarprodukt $M_i * V_j = cos(\theta)\|M_i\|\|V_j\|$ maximal wird. Das bedeutet, dass die Orthonormalität und die Längen von M_i und V_j entscheidend für die Größe des Skalarproduktes sind. Außerdem gilt, dass die kleinsten Werte entsprechend dort sein werden, wo die größte Orthogonalität und die kleinsten Längen bestehen. Ein Beispiel ist im Ausdruck 4.5 zu sehen. Alle Zeilen von M und alle Spalten von V haben die gleiche Länge, aber von allen Paaren aus M-Zeilen und V-Spalten ist nur das Paar $M_{2:}\ V_{:1}$ orthonormal. Alle andere Paare sind orthogonal. Dementsprechend hat der zweite Zeilenindex in O den größten Wert.

$$\underbrace{\begin{bmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 10 \\ 0 & 0 & 10 \end{bmatrix}}_{M} * \underbrace{\begin{bmatrix} 0 \\ 10 \\ 0 \end{bmatrix}}_{V} = \underbrace{\begin{bmatrix} 0 \\ 100 \\ 0 \\ 0 \end{bmatrix}}_{O} \tag{4.5}$$

Die Eigenschaften der Matrixmultiplikation werden bei Multi Head Attention für folgende Aufgaben benutzt:

1. **Berechnung von Schlüsseln** - erfolgt indem die erste GEMM-Schicht anhand der Anfrage $Q \in \mathbb{R}^{B \times S \times H}$ und den Schlüsseln $K \in \mathbb{R}^{B \times S \times H}$ die Ausgabeschlüsseln $O \in \mathbb{R}^{B \times S \times S}$ berechnet. Es werden die letzten zwei Dimensionen transponiert und multipliziert. Pro Sequenz werden S^2 viele Schlüssel berechnet - pro Paar aus Tokens ein Schlüssel. Der größter Schlüssel eines Tokens s_q ist im Paar mit dem Token s_j, wo der Winkel zwischen $Q_{s,q,:}$ und $K_{s,:,j}$ am kleinsten ist und die Längen von $Q_{s,q,:}$ und $K_{s,:,j}$ am größten sind. Mithilfe von Skalierung und Maskierung werden die berechneten Schlüssel stärker auf die Länge der Sequenz fokussiert und mithilfe von Softmax auf Länge 1 normiert, so dass sie eine Wahrscheinlichkeitsverteilung darstellen.

2. **Selektion von Werten** - erfolgt indem Werte aus $V \in \mathbb{R}^{B \times S \times H}$ anhand der berechneten Schlüssel selektiert werden. Die Selektion ist auf die Eigenschaft basiert, dass wenn ein auf Länge 1 normierter und nur mit positiven Werten Vektor $O \in \mathbb{R}^{1 \times 1 \times S}$ mit einer Matrix $V \in \mathbb{R}^{1 \times S \times H}$ multipliziert wird, dann gilt für die Ausgabe $Y \in \mathbb{R}^{1 \times 1 \times H}$, dass bei der Berechnung der Spalte $Y_{1,1,q} \in \mathbb{R}$ mit $Y_{1,1,q} = \sum_j O_{1,1,j} * V_{1,j,q}$ die Zeilen j der Spalte $V_{1,:,q}$ den größten Einfluss auf das Skalarprodukt haben, die mit den größten Wahrscheinlichkeiten aus dem berechneten Schlüssel O multipliziert werden. Das bedeutet, dass die größte Werte aus O die stärkste Selektivität haben.

Ein Beispiel der Selektion von Werten ist im Ausdruck 4.6 zu sehen. Der Schlüssel O hat die größte Wahrscheinlichkeit in die zweite Spalte. Entsprechend wird die zweite Zeile aus V am stärksten und in diesem Fall sogar vollständig selektiert, da $V_{1,2} = 1$.

$$\underbrace{\begin{bmatrix}0 & 1 & 0 & 0\end{bmatrix}}_{O} * \underbrace{\begin{bmatrix}1 & 0\\ 10 & 0\\ 100 & 5\\ 1000 & 6\end{bmatrix}}_{V} = \underbrace{\begin{bmatrix}10 & 0\end{bmatrix}}_{Y} \tag{4.6}$$

Der Schlüssel kann auch eine Kombination mehrerer Schlüssel sein, so dass mehrere Werte selektiert werden. In diesem Fall werden die Werte mit der jeweiligen Wahrscheinlichkeit skaliert. Ein Beispiel ist im folgenden Ausdruck zu sehen.

$$\underbrace{\begin{bmatrix}0 & 0.3 & 0.7 & 0\end{bmatrix}}_{O} * \underbrace{\begin{bmatrix}1 & 0\\ 10 & 0\\ 100 & 5\\ 1000 & 6\end{bmatrix}}_{V} = \underbrace{\begin{bmatrix}73 & 3.5\end{bmatrix}}_{Y} \tag{4.7}$$

Die Gewichtsmatrizen und die Biase für die drei Abbildungen (Anfrage, Schlüssel und Werte) sind trainierbare Parameter. Sie werden im Verlauf des Trainings in Abhängigkeit der Eingabesequenz optimiert und haben einen direkten Einfluss auf das Minimieren der Zielfunktion - wie aus der Ableitung zu sehen ist.

$$\begin{aligned}
Q &= I * W_q + B_q & \frac{\partial Q}{\partial W_q} &= \frac{\partial I * W_q + B_q}{\partial W_q} = \Delta_q^T * I & \frac{\partial Q}{\partial B_q} &= \Delta_q\\
K &= I * W_k + B_k & \frac{\partial K}{\partial W_k} &= \frac{\partial I * W_k + B_k}{\partial W_k} = \Delta_k^T * I & \frac{\partial Q}{\partial B_k} &= \Delta_k\\
V &= I * W_v + B_v & \frac{\partial V}{\partial W_v} &= \frac{\partial I * W_v + B_v}{\partial W_v} = \Delta_v^T * I & \frac{\partial Q}{\partial B_v} &= \Delta_v
\end{aligned}$$

Mit $I \in \mathbb{R}^{S \times H}$ die Eingabe, W_q, W_k und $W_v \in \mathbb{R}^{H \times H}$ die Gewichtsmatrizen und B_q, B_k und $B_v \in \mathbb{R}^{1 \times H}$ die Biase. Jede der drei Abbildungen bildet jede Zeile j der Ausgabe nur in Abhängigkeit des Tokens an Position j der Eingabesequenz. So werden keine Tokens vermischt. Das bedeutet, dass die Anfrage und das Schlüssel- bzw. Wertvokabular genau eine Kodierung pro Token haben. Die versteckte Größe der Abbildungen ist so groß wie die Kodierung pro Token aus der Worteinbettung. Diese wird in BERT und Transformer nicht vergrößert. Mithilfe der letzten Dense-Schicht der Multi Head Attention wird die endgültige Kodierung pro Token festgelegt. Das ist auch H groß.

Multi Head Attention berücksichtigt die Beziehung jeder zwei Tokens der Sequenz. Das erfolgt indem jedes Token j (Zeile j der Anfragematrix) bei der Matrixmultiplikation in Rahmen der Schlüsselberechnung mit jedem anderen Token i (Spalte i der Schlüsselmatrix) ins Skalarprodukt involviert wird, so dass für jedes dieser Paare ein Schlüssel berechnet wird. Deshalb hat O die Dimension $\mathbb{R}^{S\times S}$. Nach Abschluss des Trainings heißt das, dass in Rahmen der Wertselektion zu jeder Beziehung einen passenden Wert selektiert wird, der mit der Wichtigkeit der Beziehung in Korrelation steht. Das bedeutet, dass die Unterschiede in die Wichtigkeit der Beziehungen in die Wertunterschiede kodiert sind. Zum Beispiel in die folgende Abbildung ist zu erkennen, dass sich die Beziehungen mit und zwischen Tokens ab Stelle $j > 105$ stark von allen anderen unterscheiden. Das liegt daran, dass diese Stellen weniger Bedeutung haben. Entsprechend sind auch die selektierten Werte unterschiedlich. Innerhalb des relevanten Bereiches - $j \leq 105$ - ist zu sehen, dass auch dort Variation in die Wichtigkeit besteht, weil bestimmte Paare von Tokens mehr relevant sind als andere Paare. Es ist auch zu erkennen, dass in jedem Attention Head eine unterschiedliche Repräsentation betrachtet wird.

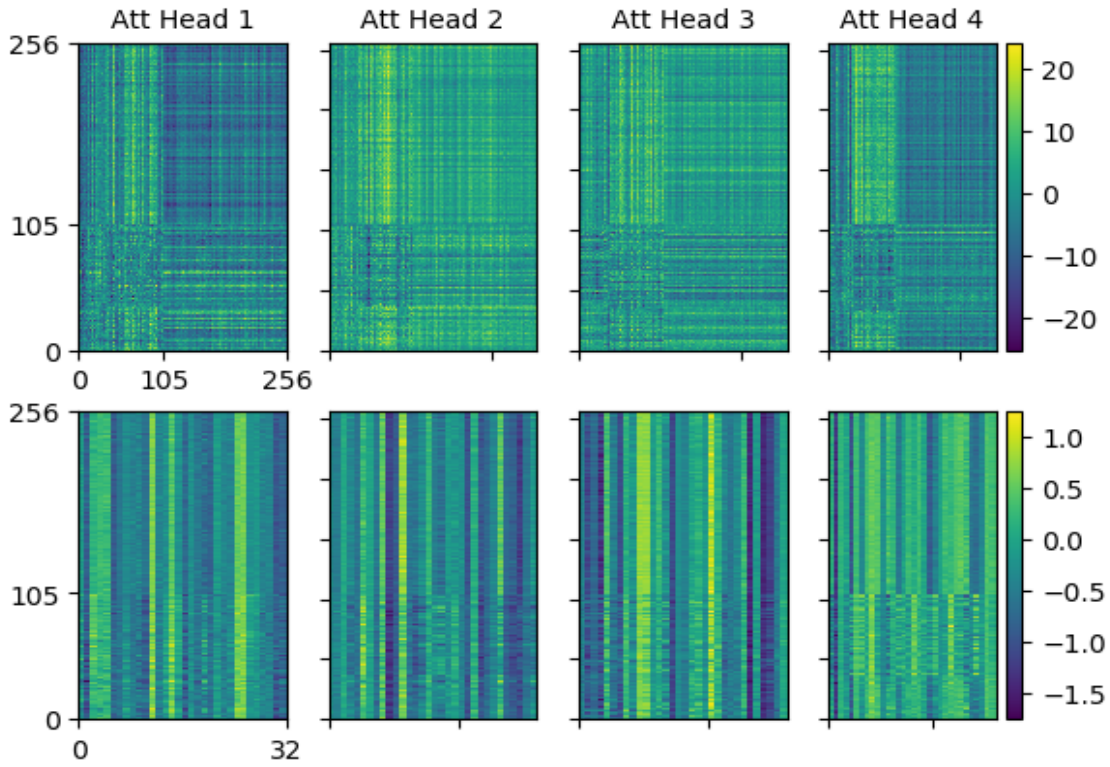

Abbildung 4.7: Nicht normierte Schlüssel (die erste Zeile) und die dazu selektierten Werte (die zweite Zeile) von 4 Attention Heads für Sequenz der Länge 256 mit 105 relevanten Stellen. Versteckte Größe ist 128 und entsprechend werden pro Attention Head 32 Werte pro Token selektiert.

Um mehr als nur eine Repräsentation der Paare von Tokens zu betrachten, wird in Multi Head Attention eine Split-Schicht integriert. Aus dieser kommt auch die Bezeichnung "Multi Head". Diese wird von dem nicht trainierbaren Parameter N (Anzahl der Multi Head Attention Heads) gesteuert. Die Split-Schicht transponiert die Anfrage, das Schlüssel- und Wortvokabular jeweils so, dass statt einer H langen Kodierung pro Token eine $\frac{H}{N}$ lange entsteht. So entstehen pro Sequenz N viele verschiedene Kodierungen pro Token. Die Vermutung ist, dass die Betrachtung mehrerer Repräsentationen die Fähigkeit des Modells verbessert, die Wichtigkeit bestimmter Paare zu erkennen. Wie aus der oberen Abbildung zu sehen ist, bestehen Unterschiede in die Werte der Attention Heads, weil jede davon anders die Wichtigkeit der Beziehungen kodiert. Nach der Aufteilung wird für jede Repräsentation separat eine Schlüsselberechnung und Wertselektion durchgeführt und im Anschluss werden alle in Rahmen der Combine-Schicht durch Transponierung wieder kombiniert.

Eine Variation in die Werte ist für die Erkennung der Unterschiede in die Wichtigkeit nicht ausreichend. Dementsprechend werden die nicht relevanten Stellen durch eine Maskierung gedämpft. Das erfolgt indem von allen Stellen, die nicht zu der Sequenz gehören, den Wert $1e5$ abgezogen wird, so dass die Softmaxfunktion diesen Stellen absolute Nullen zuweist. Das bedeutet, dass diese Stellen bei der Berechnung der Gradienten auch Nullen sind. So entsteht kein unnötiges Rauschen. Wenn eine Sequenz der Länge $\hat{S}$ kürzer als die maximale Sequenzlänge S ist, dann werden die letzten $S - \hat{S}$ viele Stellen gedämpft.

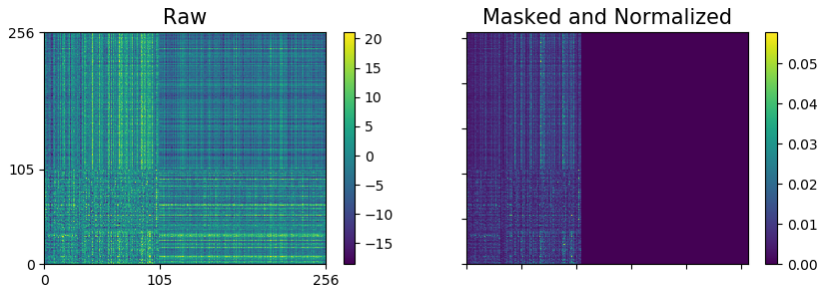

Abbildung 4.8: Einfluss der Maskierung und der Normalisierung auf die Schlüssel. Reduktion des Rauschens und mehr Eindeutigkeit bei der Wertselektion.

Multi Head Attention ist mit dem folgenden Ausdruck zu beschreiben.

$$Q = I * W_q + B_q \qquad K = I * W_k + B_k \qquad V = I * W_v + B_v$$
$$\hat{Q} = \text{split}(Q, \mathsf{N}) \qquad \hat{K} = \text{split}(K, \mathsf{N}) \qquad \hat{V} = \text{split}(V, \mathsf{N})$$
$$M' = (1 - M) * -1e5$$
$$\text{MHA}(\hat{Q}, \hat{K}, \hat{V}, M') = \text{combine}(\text{softmax}(\frac{\hat{Q} * \hat{K}^T}{\sqrt{\mathsf{H}}} + M') * \hat{V}) * W_{\text{output}}$$

4.5 Batch Normalisierung

Die Tiefe einer Schicht im Netz korreliert positiv mit den Vektorlängen ihrer Ausgabe. Das bedeutet, dass je tiefer das Netz, desto größer die Gefahr an numerische Instabilität ist. Außerdem je tiefer das Netz, desto größer ist auch die Vielfalt an verschiedenen Verteilungen, die die Ausgaben der Schichten bilden können. Das bringt die Notwendigkeit spezielle Arten von Normalisierungen in die Netzarchitektur zu integrieren, denn ohne solche ist die numerische Stabilität nur mit kleineren Lernraten zu sichern, was aber den Nachteil hat, dass das Training des Modells langsamer oder sogar unmöglich wird. Außerdem ohne spezielle Normalisierungen im Netz ist das gesamte Modell sehr sensible auf die Initialisierung der Parameter, was bedeutet, dass es schwieriger ist diese richtig zu machen. Die Auswahl des Optimierungsalgorithmus und seiner Parameter können von Normalisierungen im Netz auch profitieren und stabiler werden.

Diese Beobachtung adressieren Ioffe and Szegedy (2015) und nennen den Grund dafür Internal Covariance Shift - Verschiebung der internen Kovarianz. Um diese zu vermeiden schlagen sie den Ansatz der Batch Normalisierung vor. Die Idee ist Parameter zur Normalisierung der Ausgaben der Schichten zu erlernen. Die Technik wird häufig eingesetzt, da sie zur verbesserten Genauigkeit des Modells, robusteren Lernraten und schnellerem Training führt. Die Batch Normalisierung ist zum Bestandteil vieler NLP und Computer Vision Modelle geworden. Aufgrund ihrer Effektivität wird sie auch im BERT eingesetzt.

Die Funktionalität der Batch Normalisierung ist auf die Eigenschaft basiert, dass für jeden Vektor $X \in \mathbb{R}^S$ mit Mittelwert $\mu_X = \frac{1}{S}\sum X_i$ und Standardabweichung $\sigma_X = \sqrt{\frac{1}{S}\sum (X_i - \mu_X)^2}$ gilt, dass die Transformation $\hat{X} = \frac{(X-\mu_X)}{\sigma_X}$ die Werte von X so abbildet, dass $\mu_{\hat{X}} = 0$ und $\sigma_{\hat{X}} = 1$. Wenn die Transformation um eine Multiplikation mit einem Skalar γ und eine Addition mit einem Skalar β erweitert wird, dann gilt für $\hat{X} = \gamma\frac{(X-\mu_X)}{\sigma_X} - \beta$, dass $\mu_{\hat{X}} = \beta$ und $\sigma_{\hat{X}} = \gamma$.

Die Aussage über den Mittelwert ist sehr hilfreich, denn durch eine Addition mit β kann den Mittelwert von $\hat{X}$ kontrolliert werden. Der Beweist ist wie folgt:

$$
\begin{aligned}
\mu_{\hat{X}} &= \frac{1}{S}\sum \frac{\gamma\,(X_i - \mu_X)}{\sigma_X} + \beta \\
&= \frac{1}{S}\left(\sum \frac{\gamma X_i}{\sigma_X} - \sum \gamma \frac{\frac{1}{S}\sum X_i}{\sigma_X} + \sum \beta\right) \\
&= \frac{1}{S}\left(\sum \frac{\gamma X_i}{\sigma_X} - \sum \frac{\gamma X_i}{\sigma_X} + S\beta\right) \\
&= \beta
\end{aligned}
$$

Der folgende Beweist bestätigt die Aussage über den Einfluss von γ auf die Standardabweichung. Das bedeutet, dass eine einzige Multiplikation hinreichend ist, um die Standardabweichung von $\hat{X}$ zu steuern.

$$\begin{aligned}
\sigma_{\hat{X}} &= \sqrt{\frac{1}{S}\sum\left(\frac{\gamma\left(X_i-\mu_X\right)}{\sigma_X}+\beta-\mu_{\hat{X}}\right)^2} \\
&= \sqrt{\frac{1}{S}\sum\left(\frac{\gamma^2\left(X_i-\mu_X\right)^2}{\sigma_X^2}+2\frac{\gamma\left(X_i-\mu_X\right)}{\sigma_X}\beta+\beta^2-2\left(\gamma\frac{(X-\mu_X)}{\sigma_X}-\beta\right)\mu_{\hat{X}}+\mu_{\hat{X}}^2\right)} \\
&= \sqrt{\frac{\gamma^2}{S\sigma_X^2}\sum\left(X_i-\mu_X\right)^2+\frac{2\gamma\beta}{S\sigma_X}\sum\left(X_i-\mu_X\right)+\beta^2-\frac{2\gamma\mu_{\hat{X}}}{S\sigma_X}\sum\left(X_i-\mu_X\right)+\frac{1}{S}\sum 2\beta\mu_{\hat{X}}+\mu_{\hat{X}}^2} \\
&= \sqrt{\frac{\gamma^2}{\sigma_X^2}\sigma_X^2+\frac{2\gamma\beta\mu_X}{\sigma_X}-\frac{2\gamma\beta\mu_X}{\sigma_X}+\beta^2-\frac{2\gamma\mu_{\hat{X}}\mu_X}{S\sigma_X}+\frac{2\gamma\mu_{\hat{X}}\mu_X}{S\sigma_X}+2\beta\mu_{\hat{X}}+\mu_{\hat{X}}^2} \\
&= \sqrt{\gamma^2+\beta^2+2\beta\mu_{\hat{X}}+\mu_{\hat{X}}^2} = \sqrt{\gamma^2+\left(\beta-\mu_{\hat{X}}\right)^2} = \sqrt{\gamma^2+(\beta-\beta)^2} = \gamma
\end{aligned}$$

Die Batch-Normalisierung erlaubt die Verteilung von $\hat{X}$ durch die beiden Parameter γ und β zu steuern. Das Ziel ist diese so zu steuern, dass die Ausgaben der Schichten auf Verteilungen $\mathcal{N}(\mu,\sigma)$ mit μ möglichst nah an 0 und σ positiv und klein abgebildet werden. Dadurch hat jede $x \in \hat{X}$ eine möglichst kleine Abweichung von 0. So führt jede Skalarmultiplikation zu kleineren Skalarwerten und jede Matrixmultiplikation zu Vektoren mit kleineren Vektorlängen. Der Beweis ist in die folgenden Grenzwerte zu sehen.

$$\begin{aligned}
\lim_{\sigma_X\to\infty} Y * \frac{(X-\mu_X)}{\sigma_X} &= \lim_{\sigma_X\to\infty}\sum Y_j * \frac{(X_j-\mu_X)}{\sigma_X} = 0 \\
\lim_{\sigma_X\to 0} Y * \frac{(X-\mu_X)}{\sigma_X} &= \lim_{\sigma_X\to 0}\sum Y_j * \frac{(X_j-\mu_X)}{\sigma_X} = \lim_{\sigma_X\to 0}\sum Y_j * \frac{(\mu_X-\mu_X)}{\sigma_X} = 0
\end{aligned}$$

Bei Skalarmultiplikation zwischen $\hat{X}$ und ein $Y \in \mathbb{R}^S$ gilt, dass je größer $|\sigma_X|$ desto kleiner wird das Ergebnis, denn durch die Division wird jeder Summand auch kleiner. Wenn $|\sigma_X|$ sehr klein wird, dann haben alle Werte eine sehr kleine Abweichung von dem Mittelwert, was bedeutet, dass die Differenz nah an 0 wird. Diese Eigenschaft der Batch-Normalisierung bringt den Vorteil, dass die Gradienten auch kleine Vektorlängen haben, numerisch stabil sind und dadurch werden beim Training größere Lernraten ermöglicht.

Bezüglich der Wahl von γ und β ist vielleicht zu vermuten, dass die triviale Wahl wie $\gamma = 0$ und $\beta = 1$ gut ist. Allerdings diese und jede konstante Wahl trifft die Entscheidung, dass es bekannt ist, welcher Einfluss jeder Teil des Netzes auf die Optimierung hat, weil nur mit solchem Wissen diese Parameter bestimmt werden können.

Es ist aber meistens kein a-priori Wissen über die Verteilung der Ausgaben der einzelnen Schichten verfügbar und dementsprechend werden die Parameter γ und β während des Trainings optimiert. Die Berechnungsvorschrift der Gradienten der Batch-Normalisierung ist in die folgenden Abbildungen zu sehen. Zuerst wird gezeigt, dass die Ableitung des Zählers bezüglich der Eingabe eine Subtraktion ist. Das ist vorteilhaft zu wissen, denn diese ist $\mathcal{O}(n)$ und das spart unnötige Berechnungen bei der Optimierung.

$$\begin{aligned}
\sum_{\hat{X}_j} \Delta_j \frac{\partial \gamma \left(x_j - \mu_X\right)}{\partial x_i} &= \gamma \left(\Delta_i \frac{\partial \left(x_i - \mu_X\right)}{\partial x_i} + \sum_{j \neq i} \Delta_j \frac{\partial \left(x_j - \mu_X\right)}{\partial x_i} \right) \\
&= \gamma \left(\Delta_i - \frac{\Delta_i}{S} - \sum_{j \neq i} \frac{\Delta_j}{S} \right) \\
&= \gamma \left(\Delta_i - \frac{1}{S} \sum_j \Delta_j \right) \\
\Delta \frac{\partial \gamma \left(X - \mu_X\right)}{\partial X} &= \gamma \left(\Delta - \frac{1}{S} \sum_j \Delta_j \right)
\end{aligned}$$

Die Standardabweichung wird auch abgeleitet und es ergibt sich die folgende Gleichung. Es ist zu sehen, dass die Standardabweichung bei der Berechnung der Gradienten wiederverwendet werden kann, was zur Vermeidung von unnötigen und doppelten Berechnungen führt.

$$\begin{aligned}
\frac{\partial \sigma_X}{\partial x_i} &= \frac{\partial \sqrt{\frac{1}{S} \sum \left(x_i - \mu_X\right)^2}}{\partial x_i} \\
&= \frac{1}{2} \frac{1}{\sigma_X} \frac{1}{S} \left(\frac{\partial \left(x_i - \mu_X\right)^2}{\partial x_i} + \sum_{j \neq i} \frac{\partial \left(x_j - \mu_X\right)^2}{\partial x_i} \right) \\
&= \frac{1}{2} \frac{1}{\sigma_X} \frac{1}{S} \left(2 \left(x_i - \mu_X\right) \left(1 - \frac{1}{S}\right) - \sum_{j \neq i} 2 \left(x_j - \mu_X\right) \frac{1}{S} \right) \\
&= \frac{1}{2} \frac{1}{\sigma_X} \frac{1}{S} \left(2 \left(x_i - \mu_X\right) - \frac{2}{S} \sum x_j - \mu_X \right) \\
&= \frac{1}{2} \frac{1}{\sigma_X} \frac{1}{S} \left(2 \left(x_i - \mu_X\right) - 2\mu_X + 2\mu_X \right) \\
&= \frac{1}{\sigma_X} \frac{1}{S} \left(x_i - \mu_X\right)
\end{aligned}$$

Mithilfe der zwei oben formulierten Ableitungen kann man die Gesamtableitung bezüglich der Eingabe formulieren. Es ist zu sehen, dass diese sich auch effizient berechnet lässt, da die Standardabweichung und die Ausgaben der Schicht wiederverwendet werden. Auch hier profitiert der Gradient von numerischer Stabilität auf-

grund der Eigenschaft, dass eine sehr kleine bzw. sehr große Standardabweichung zu Vektoren mit kleinen Vektorlängen führt.

$$\begin{aligned}
\frac{\partial \hat{X}}{\partial x_i} &= \sum_{\hat{X}_j} \Delta_j \frac{\frac{\partial \gamma (x_j - \mu_X)}{\partial x_i} \sigma_X - \frac{\partial \sigma_X}{\partial x_i} \gamma (x_j - \mu_X)}{\sigma_X^2} \\
&= -\frac{\gamma \sigma_X \left(\Delta_i - \frac{1}{S} \sum \Delta_j \right)}{\sigma_X^2} - \frac{\gamma}{\sigma_X^2} \sum_{\hat{X}_j} \Delta_j \frac{\partial \sigma_X}{\partial x_i} (x_j - \mu_X) \\
&= \frac{\gamma \left(\Delta_i - \frac{1}{S} \sum \Delta_j \right)}{\sigma_X} - \frac{\gamma}{\sigma_X^2} \sum_{\hat{X}_j} \Delta_j \frac{1}{\sigma_X} \frac{1}{S} (x_i - \mu_X) (x_j - \mu_X) \\
&= \frac{\gamma \left(\Delta_i - \frac{1}{S} \sum \Delta_j \right)}{\sigma_X} - \frac{\gamma}{S \sigma_X^3} (x_i - \mu_X) \sum_j \Delta_j (x_j - \mu_X) \\
&= \frac{\gamma \left(\Delta_i - \frac{1}{S} \sum \Delta_j \right)}{\sigma_X} - \frac{1}{S \sigma_X} \frac{\gamma (x_i - \mu_X)}{\sigma_X} \sum_j \Delta_j \frac{(x_j - \mu_X)}{\sigma_X} \\
\frac{\partial \hat{X}}{\partial X} &= \frac{\gamma \left(\Delta - \frac{1}{S} \sum \Delta_j \right)}{\sigma_X} - \frac{1}{S \sigma_X} \frac{\gamma (X - \mu_X)}{\sigma_X} \left(\Delta^T * \frac{X - \mu_X}{\sigma_X} \right)
\end{aligned}$$

Die Gradienten der Parameter γ und β sind in folgende Gleichungen zu sehen. Der Mittelwert der Ausgabe wird in Richtung des Gradienten der danach folgenden Schicht geschoben. Für die Standardabweichung gilt auch, dass diese am stärksten davon beeinflusst wird.

$$\frac{\partial \hat{X}}{\partial \gamma} = \Delta * \frac{(X - \mu_X)}{\sigma_X}$$

$$\frac{\partial \hat{X}}{\partial \beta} = \sum \Delta_j$$

Der einzige Metaparameter der Batch-Normalisierung ist die Auswahl an Achsen, über die die Aggregation stattfindet. In Computer Vision wird häufig die Achse der Kanäle ausgewählt. Beim BERT wird die Achse der Kodierung der Tokens benutzt - die letzte Achse.

4.6 Dropout

Auf Gradientenabstiegsverfahren basierte erste Ordnung Optimierungsalgorithmen können im Verlauf des Trainings zu einem Sättigungspunkt kommen, so dass ab diesem keine Verbesserung des Modells mehr möglich ist. Wenn die Trainingsdauer viele Iteration beträgt oder der Datensatz klein ist, kann es dazu kommen, dass das Modell die Erkennung der Trainingsdaten perfektioniert, aber gleichzeitig nicht akzeptable Resultate beim Testen auf unbekannte Daten zeigt. Wenn das passiert, dann spricht man vom Overfitting. Eine Möglichkeit das zu vermeiden ist der Ansatz von Srivastava et al. (2014) mit Dropout Schichten. Diese, wie der Name schon andeutet, basieren auf die Idee Teile der Ausgaben zu verwerfen bzw. auszuschalten. Das erfolgt indem $\alpha\%$ Indices der Eingabe gemäß einer Wahrscheinlichkeitsverteilung ausgewählt werden. Die Werte der Eingabe, die den Indizes entsprechen, werden auf 0 gesetzt. Die Dropout hat die folgende Berechnungsvorschrift mit $I \in \mathbb{N}^{\alpha * S}$ und $\alpha \in [0, 1]$.

$$\begin{aligned} \textsf{Droput}(X) &= \hat{X} & j \in I \implies \hat{X}_j = 0 \land j \notin I \implies \hat{X}_j = X_j \\ \frac{\partial \textsf{Dropout}(X)}{\partial X} &= \hat{\Delta} & j \in I \implies \hat{\Delta}_j = 0 \land j \notin I \implies \hat{\Delta}_j = \Delta_j \end{aligned}$$

Die Auswahl $\alpha\%$ viele Indices der Eingabe und die Setzung dieser auf 0 erfolgt in jedem Trainingsschritt. So werden in jede Iteration verschiedene Versionen von X durch das Netz propagiert. Das bringt im Training mehr Variation und hilft dem Netz Unvollständigkeit der Daten zu erkennen. Es wird also jedes Mal eine unterschiedliche Repräsentation der Daten gesehen und anhand deren die Parameter optimiert. So ist die Wahrscheinlich kleiner, dass die Gradienten jedes Mal die Optimierung in die gleiche Richtung und mit der gleichen Intensität verschieben. Das bedeutet, dass auch die Wahrscheinlichkeit an Overfitting reduziert ist.

Die Auswahl einer Wahrscheinlichkeitsverteilung für die Indexberechnung zählt zu dem Metaparameter von Dropout. In die ursprüngliche Version wird eine Bernoulli Verteilung verwendet, aber erwähnt, dass die Auswahl von Normalverteilung zu besseren Resultaten. Der zweite Metaparameter der Dropout-Schicht ist α. Der steuert die Anzahl an verworfenen Stellen in die Ausgabe. Wenn er sehr groß ist, dann werden zu viele Stellen der Eingabe auf 0 gesetzt, was bedeutet, dass dadurch die Datenrepräsentation sehr unvollständig und gefälscht wird. Das kann dazu führen, dass das Modell Daten erlernt, die nicht mehr repräsentativ für die Aufgabe sind. Dementsprechend wird der Parameter klein gehalten. Im BERT ist er auf 0.1 gesetzt. Aus der Ableitung der Dropout-Schicht ist zu sehen, dass nur die nicht verworfenen Indizes in die Optimierung involviert bleiben. Es ist wichtig, dass die Wahrscheinlichkeitsverteilung jedes Mal verschiedene Indizes auswählt, so dass nicht immer die gleichen Stellen optimiert werden.

4.7 Aktivierungsfunktionen

Die Integration nicht linearer Funktionalität in Netzarchitektur wird unter der Annahme gemacht, dass dadurch Abhängigkeiten beim Optimieren erlernt werden können, die zur verbesserten Genauigkeit des Modells führen, denn sehr komplexe Abhängigkeiten durch Linearität nicht definierbar sind. Die Auswahl an nicht linearen Funktionen kann umfangreich sein und jede Funktion hat ihre Vor- und Nachteile. Bezüglich BERT ist es erforderlich, dass die benutzen Aktivierungsfunktionen unabhängig der Netztiefe sind, was bedeutet, dass Kandidaten wie Sigmoid ausgeschlossen sind. Außerdem sollen alle Stellen der Aktivierung berücksichtigt werden und nicht nur diese mit positiven Skalarwerten. Somit ist ReLU auch ausgeschlossen. Die Funktion soll über die Eigenschaft verfügen, nur bestimmte Intervalle des Definitionsbereichs stärker zu betonen. Und die Stärke der Betonung soll konfigurierbar sein. Die im BERT zum Einsatz kommende Aktivierungsfunktion erfüllt zum Großteil diese Anforderungen. Es wird die GeLU (Gaussian Linear Unit) von Hendrycks and Gimpel (2016) benutzt. Die Berechnungsvorschrift dieser ist in die folgende Gleichung zu sehen.

$$\textbf{GeLU}(x) = 0.5x + \tanh\left(\sqrt{\frac{2}{\pi}}\left(x + 0.044715x^3\right)\right)$$

Ein Einblick in dem Funktionsverlauf von GeLU und den anderen genannten Aktivierungsfunktionen ist in Abbildung 4.9 zu sehen.

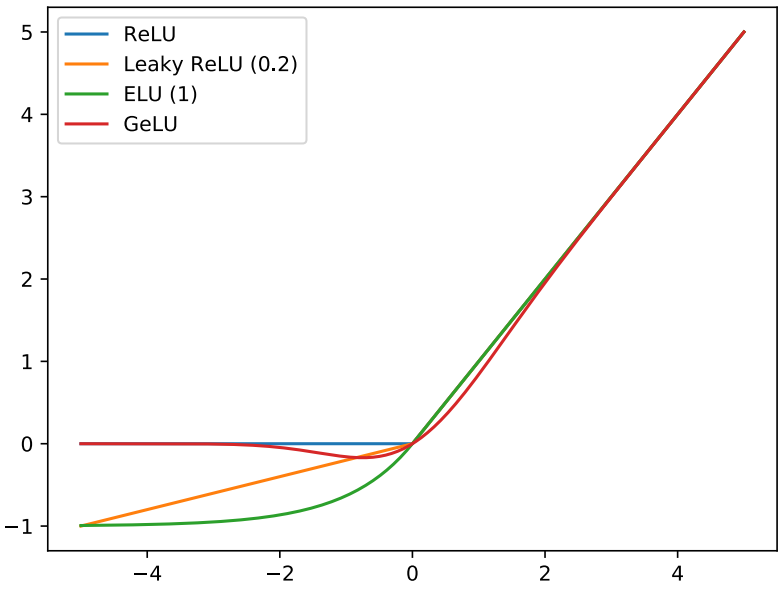

Abbildung 4.9: Funktionsgraphen von ReLU, Leaky ReLU, ELU und GeLU

Es ist zu sehen, dass GeLU für positive Eingabewerte nah an die anderen dargestellten Funktionen kommt. Im Intervall $[2, \infty]$ sind die Unterschiede kaum erkennbar.

Der Unterschied von GeLU ist allerdings im negativen Eingabebereich, da sie das Intervall $[-2, 0]$ stark und das Intervall $[-\infty, -2]$ schwach betont. Es kann vorteilhaft sein, wenn nur bestimmte Intervalle negativer Werte gedämpft werden, denn so werden andere Intervalle intensiver in die Optimierung involviert. Laut den Autoren ist die GeLU fähig das Training des Modells zu beschleunigen. Die Berechnungsvorschrift der Ableitung ist wie folgt definiert.

$$\frac{\partial \textbf{GeLU}(x)}{\partial x} = 0.5 + \left(1 - \tanh\left(\sqrt{\frac{2}{\pi}}\left(x + 0.044715x^3\right)\right)^2\right)\left(\sqrt{\frac{2}{\pi}} + 3 * 0.0044715x^2\right)$$

Ein weiterer Vorteil von GeLU ist, dass die Ableitung nicht für jede Eingabe konstant, sondern im Intervall $[-2.2, 2.2]$ von der Eingabe abhängig ist. So wird nicht nur das Vorzeichen, sondern auch der Wert der Eingabe betrachtet. Diese Beobachtung ist in Abbildung 4.10 zu sehen. Die drei anderen Funktionen sind im positiven Bereich konstant. Im negativen Bereich haben nur ELU und GeLU eine Abhängigkeit von der Eingabe. GeLU hat einen besonderen Einfluss auf die Optimierung, weil sie positive von negativen Aktivierungen nicht nur anhand des Vorzeichens unterscheidet, sondern auch des Wertes.

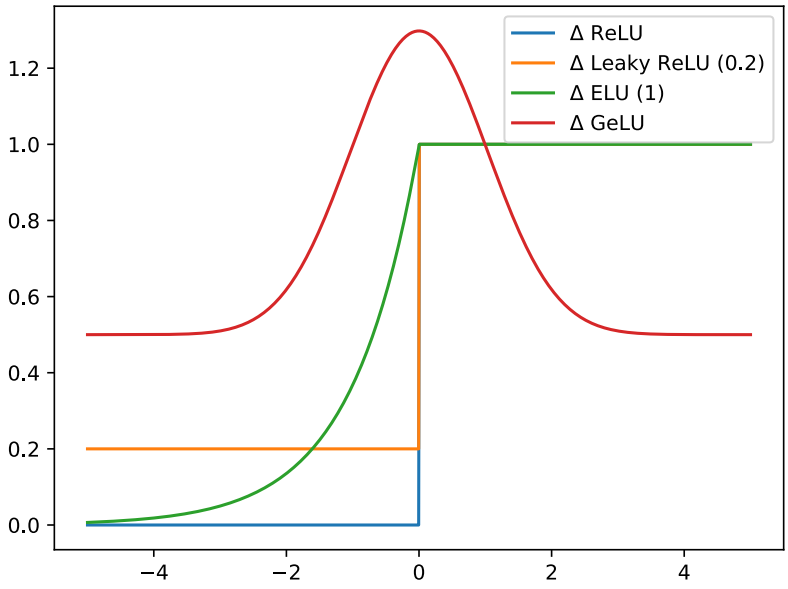

Abbildung 4.10: Ableitungen von ReLU, LeakyReLU, ELU und GeLU

4.8 Residual Verbindungen

Ein gängiger Ansatz zur Verbesserung der Genauigkeit von Deep Learning Modellen besteht in dem trivialen Vorgang das Netz tiefer zu gestalten - mehrere Schichten einzufügen. Allerdings je tiefer das Netz, desto langsamer und schwieriger wird das Training. Einer der Gründe dafür ist, dass die im Backpropagation geleiteten Gradienten von Schicht zur Schicht teilweise verschwinden oder ungünstig geändert werden können. Ein weiterer Grund ist, dass die Optimierung auf bestimmte Teile des Netzes nicht genug fokussiert sein kann. Eine Möglichkeit das zu lösen schlagen He et al. (2015) mit dem Ansatz von Residual Connections vor. Ihr Ansatz ist auf die Idee basiert, dass bestimmte Schichten im Netz zu mehr als einer anderen Schicht eine Verbindung benötigen. So können diese Schichten Teile des Netzes überspringen. Einblick in die Architektur von Residual Netzen ist in Abbildung 4.11 zu sehen.

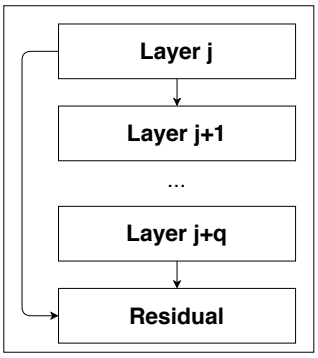

Abbildung 4.11: Residual Netz mit $q \geq 1$

Der Ansatz ist weitverbreitet und wird sowohl in Computer Vision Modelle wie YOLO v3, MobileNet v2 als auch in NLP Modelle wie BERT und Transformer eingesetzt. Die Residual Schicht verbindet zwei Schichten, die nicht unbedingt aufeinander folgend sind, und auf die Ausgaben dieser zwei Schichten wendet eine Funktion an. In der Regel ist diese eine Addition. Allerdings es kann auch ein anderer Funktionstyp sein. Die Berechnungsvorschrift sieht wie folgt aus.

$$\textsf{Residual}(X_j, X_{j+q}) = X_j + X_{j+q}$$

Die Annahme ist, dass die Residual anhand der Addition die Auswirkung der Ausgabe X_j verstärkt, denn diese nicht nur einen direkten Einfluss auf X_{j+1}, sondern auch auf die Schicht nach der Residual hat. In Computer Vision führt der Ansatz in Modelle mit Convolutional Schichten zu verbesserten Genauigkeiten, da bestimmte Repräsentationen der Eingabebilder mehrfach betrachtet oder mit anderen vermischt werden, so dass neue entstehen. Die Residuals sind auch in BERT innerhalb der Encoder-Schicht integriert, denn die Vermutung ist, dass derselbe Effekt mit den Repräsentationen der Eingabesequenzen erreicht wird.

4.9 Optimieren

Ziel der Optimierung eines Deep Learning Netzes ist es, die Parameter θ so einzustellen, dass eine Zielfunktion F minimiert wird, die den Unterschied zwischen den Ausgaben des Netzes $\hat{Y}$ und der gewünschten Ausgaben Y misst.

$$\underset{\theta}{\operatorname{argmin}} F(\hat{Y}, Y|\theta)$$

Im Rahmen der Optimierung wird versucht, ein globales Minimum zu finden. Das ist für empirische Daten ein ungelöstes Problem, wenn man nicht den kompletten Parameterraum durchsuchen will, was sehr aufwändig oder unmöglich sein kann. Außerdem nicht jede Funktion hat so eine Stelle - zum Beispiel $f(x) = x^3$. Deshalb sucht man nach einem lokalen Minimum und nimmt an, dass es sehr ähnlich dem global Minimum ist. Eine Funktion kann aber auch mehrere lokale Minimums haben - wie zum Beispiel $f(x) = sin(x)$. Es besteht auch kein a-priori Wissen über die Minimums oder Maximums der Ausgabefunktion des Netzes. Es wird die Annahme getroffen, dass es mindestens ein lokales Minimum existiert und dies zu einer akzeptablen Minimierung der Zielfunktion führt. Die Suche nach solchen Stellen wird unter anderem mit dem Gradientenabstiegsverfahren gemacht. Das ist auf Calculus basiert und zwar auf die mathematische Eigenschaft, dass die Ableitung $f' : \mathbb{R} \to \mathbb{R}$ einer differenzierbaren Funktion $f : \mathbb{R} \to \mathbb{R}$ die Stärke des Verhältnisses zwischen Änderungen in f und Änderungen in $x \in \mathbb{R}$ misst.

$$f'(x) = \lim_{\delta \to 0} \frac{f(x+\delta) - f(x)}{x - \delta - x} = \lim_{\delta \to 0} \frac{f(x+\delta) - f(x)}{\delta}$$

Der Ableitungswert $f'(x)$ an einer Stelle $x \in \mathbb{R}$ gibt an, wie stark sich die Stellen, die größer als x und sehr nah an x sind, bzgl. ihrer Funktionswerte von dem Funktionswert $f(x)$ unterscheiden. Zum Beispiel für sehr schnell wachsende Funktionen wie $f(x) = e^x$ zeigt die Ableitung $f'(x) = e^x$, dass die Größe der Funktionswerte sehr stark auf Eingabeänderungen reagiert. Für langsam wachsende Funktionen wie $f(x) = x$ mit $f'(x) = 1$ ist genau umgekehrt. Das bedeutet, dass an die Nullstellen der Ableitung das Wachstum bzw. Absteigen von f minimal ist. Es können mehrere solche Stellen vorhanden sein und diese werden Sättigungspunkte oder lokale Extremwerte genannt.

Wenn die Stelle $\hat{x} \in \mathbb{R}$ mit $\operatorname{argmin}_x f(x) = \hat{x}$ einer Funktion f gesucht wird und ein beliebiges $x \in \mathbb{R}$ eingegeben ist, dann wird der Funktionswert $f(x)$ durch die Verschiebung der Eingabe x um $x = x - \alpha f'(x)$ mit $\alpha \in [0, 1]$ in die Gegenrichtung des Steigens bzw. Absteigens bewegt, denn es gilt $f'(x) > 0$, wenn $f(x)$ steigt, und $f'(x) < 0$, wenn $f(x)$ absteigt. Das bedeutet, dass wenn $f(x)$ steigt, werden x und $f(x - \alpha f'(x))$ verkleinert, und umgekehrt. Wenn $f'(x)$ größer als x ist, wird es zu einem Vorzeichenwechsel kommen. Dementsprechend wird $f'(x)$ mit einem kleinen Wert α ska-

liert. Allerdings je kleiner α ist, desto mehrere Änderungen werden benötigt, um an einem lokalen Minimum zu kommen. Die Art und Weise wie die Änderungen von x vorgenommen werden ist entscheidend für die Suche nach dem lokalen Minimum und dafür gibt es verschiedene Algorithmen. Diese werden Optimierer genannt. Es gibt allerdings keinen Perfekten, der zu allen Modellen passt. Meistens ist der Trade-Off zwischen Geschwindigkeit und numerischer Stabilität, was bedeutet, dass die Größe des Modells auch einen Einfluss auf die Geschwindigkeit des Optimierens hat.

Zu den gängigen Optimierern zählen unter anderem das stochastische Gradientenabstiegsverfahren (SGD), das Momentum SGD, die Root Mean Squared (RMS) und die Adaptive Momentum Estimation (ADAM). Im BERT wird eine modifizierte Version von ADAM eingesetzt, die um einen Zerfall von θ erweitert ist und gleichzeitig die Größen $\hat{\mu_j}$ und $\hat{\sigma_j}$ weglässt. Alle fünf Algorithmen ändern die Parameter θ mithilfe der Ableitung $\partial\theta$ gemäß der folgenden Berechnungsvorschriften.

$$\textbf{SGD}(\alpha) : \theta_j = \theta_{j-1} - \alpha * \partial\theta_j$$

$$\textbf{Momentum SGD}(\alpha, \beta) : \theta_j = \theta_{j-1} - \Delta_j \quad \Delta_j = \beta\Delta_{j-1} + \alpha\partial\theta_j$$

$$\textbf{RMS}(\alpha, \gamma) : \theta_j = \theta_{j-1} - \Delta_j \quad \Delta_j = \frac{\alpha}{\sqrt{v_j}}\partial\theta_j \quad v_j = \gamma v_{j-1} + (1-\gamma)\left(\partial\theta_j\right)^2$$

$$\begin{aligned}\textbf{ADAM}(\alpha, \beta_1, \beta_2, \epsilon, i) : \theta_j &= \theta_{j-1} - \Delta_j \quad \Delta_j = \alpha\frac{\hat{\mu_j}}{\sqrt{\hat{\sigma_j}} + \epsilon} \\ \hat{\mu_j} &= \frac{\mu_j}{1-\beta_1^{i+1}} \qquad \hat{\sigma_j} = \frac{\sigma_j}{1-\beta_2^{i+1}} \\ \mu_j &= \beta_1\mu_{j-1} + (1-\beta_1)\partial\theta_j \qquad \sigma_j = \beta_2\sigma_{j-1} + (1-\beta_2)\left(\partial\theta_j\right)^2\end{aligned}$$

$$\begin{aligned}\textbf{BERT ADAM}(\alpha, \beta_1, \beta_2, \epsilon, \gamma) : \theta_j &= \theta_{j-1} - \Delta_j \quad \Delta_j = \alpha\left(\frac{\hat{\mu_j}}{\sqrt{\hat{\sigma_j}} + \epsilon} + \gamma\theta_{j-1}\right) \\ \mu_j &= \beta_1\mu_{j-1} + (1-\beta_1)\partial\theta_j \qquad \sigma_j = \beta_2\sigma_{j-1} + (1-\beta_2)\left(\partial\theta_j\right)^2\end{aligned}$$

Der Vorteil von SGD ist, dass er sich direkt in Richtung des lokalen Extremums orientiert und kann schnell konvergieren, da die Parameter so stark wie die Groβe des Ableitungswertes geändert werden können. Das macht ihn aber sehr sensibel gegenüber dem Vorzeichen und Größe von $\partial\theta$ und der Größe von α. Wenn sich beim Training das Vorzeichen der Gradienten in jeder Iteration ändert, kann es zu einer ständigen Oszillation zwischen positiven und negativen Werten kommen. Das ist problematisch, weil die lokal optimale Stelle immer wieder übersprungen wird. Außerdem ist die Zahlengenauigkeit im Rechner begrenzt und der SGD-Algorithmus kann sehr oft zu Zahlenübergängen kommen, wenn das Modell sehr viele Schichten hat,

weil die Gradienten von dem Optimierer nicht normiert werden. SGD ist simple, effizient und einfach zu implementieren, aber wird in große Modells nicht benutzt.

Eine Möglichkeit zur Steuerung der Sensibilität gegenüber der Oszillation von Ableitungen bietet der Momentum SGD vor. Er speichert jede Parameteränderung von θ und interpoliert zwischen dieser und der aktuellen Ableitung. Durch den Parameter β wird gesteuert, wie stark der Einfluss der letzten Parameteränderung auf die aktuelle ist. So wird die Wahrscheinlichkeit minimiert, dass die lokal optimale Stelle übersprungen wird, weil die Richtung jeder Parameteränderung aus allen bisherigen ermittelt wird und somit eine bessere Repräsentativität hat. Momentum SGD kann aber langsam werden, wenn sich die Vorzeichen der Ableitungen aufeinander folgender Iterationen unterscheiden, aber ihre Werte ähnlich sind, denn so werden sich die Werte gegenseitig ausschließen. Der Nachteil des Algorithmus liegt auch daran, dass er langsam die Richtung und die Geschwindigkeit der Parameteränderung wechselt. Wenn die Standardkombination aus $\beta = 0.9$ und eine Lernrate α nah an 0 benutzt wird und die Optimierung zu Stellen kommt, wo sich die Ableitung stark von diesen der vorherigen Iterationen unterscheidet, benötigt der Algorithmus viele Iterationen, um die Richtung und die Geschwindigkeit zu korrigieren, weil die aktuelle Ableitung eine schwächere Auswirkung auf die Parameteränderung hätte als die gespeicherte Parameteränderung.

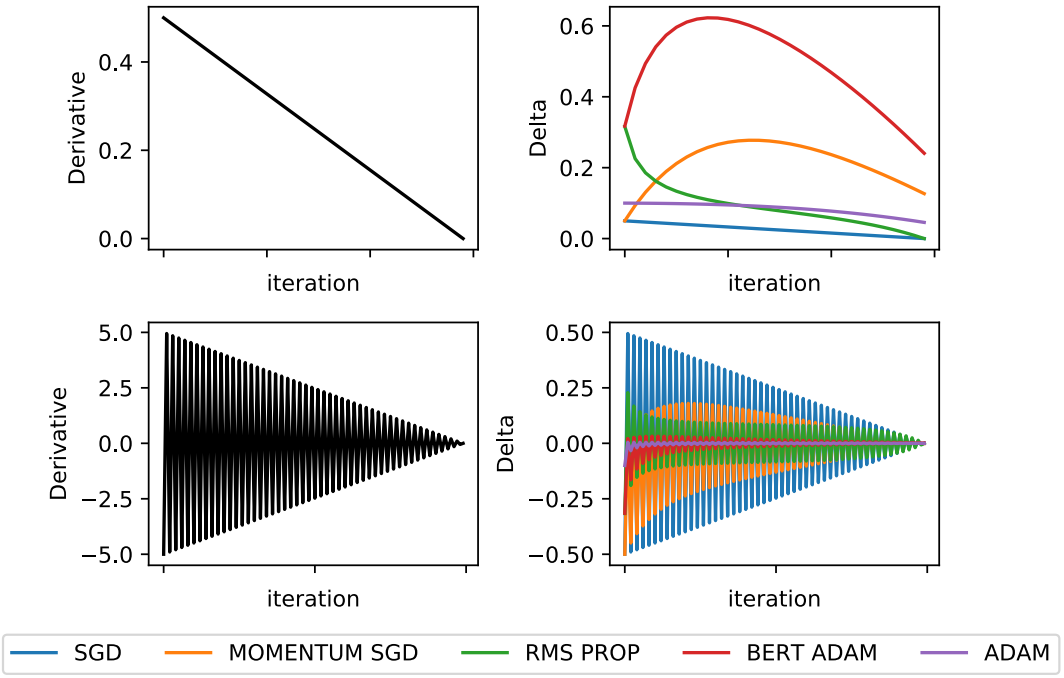

Abbildung 4.12: Auswirkung der Oszillation der Ableitung auf die Deltas der fünf vorgestellten Optimierer

Wenn sich der Betrag der Ableitung in jeder Iteration linear reduziert, dann bewegt sich die Optimierung linear schnell in Richtung des lokalen Optimums. Je kleiner die Ableitung, desto kleiner sollen die Parameteränderungen sein. Wie es aus der Abbildung 4.12 zu sehen ist, benötigt der Momentum SGD viele Iterationen um die Richtung und die Geschwindigkeit zu korrigieren. Außerdem ist er trotz der Interpolation stark sensibel gegen Oszillationen der Ableitung. Die Sensibilität gegenüber Geschwindigkeits- und Richtungswechsel wird von Root Mean Squared (RMS) und den beiden Varianten von Adaptive Momentum Estimation (ADAM) adressiert. Das ist in Abbildung 4.12 zu erkennen.

Im RMS hat die Parameteränderung eine dynamische Lernrate von $\frac{\alpha}{\sqrt{v_j}}$ und diese korreliert negativ mit der Größe der Ableitung, weil der Wert v_j negativ mit der Größe der Ableitung korreliert. Das führt zu einer Normierung der Geschwindigkeits- und Richtungsänderung pro Iteration. So werden diese Änderungen sensibler gegenüber der Monotonie der Ableitung.

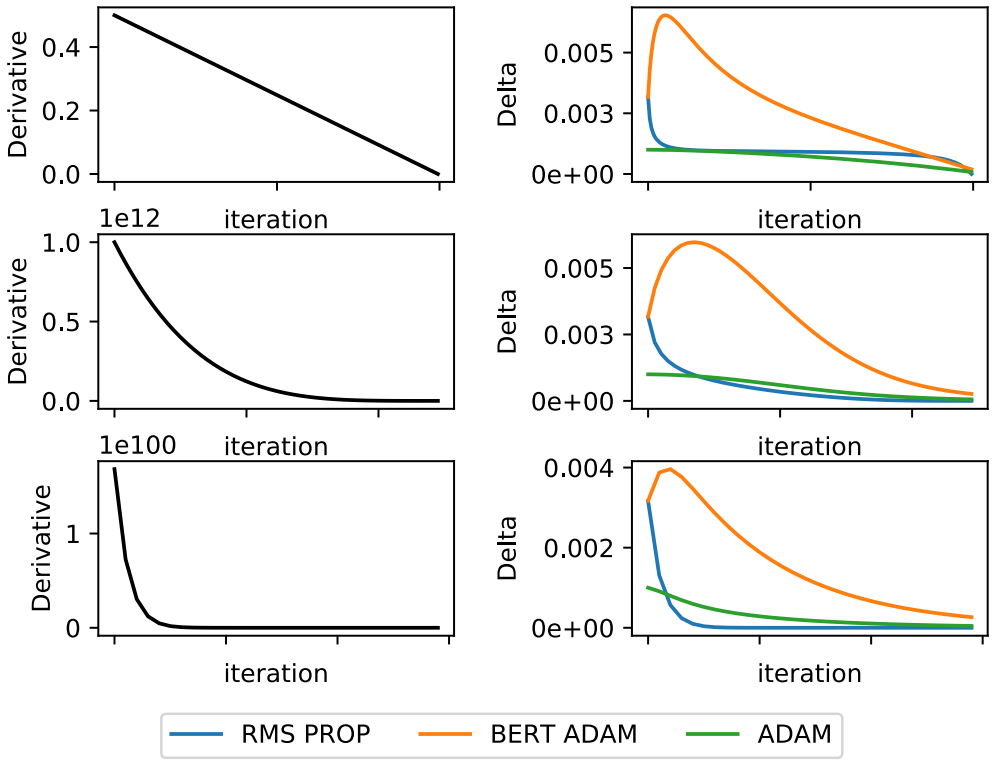

Abbildung 4.13: Auswirkung der Monotonie der Ableitung auf die Parameteränderung. Positive Korrelation in RMS Prop und ADAM. BERT ADAM zeigt eine schwache Korrelation.

Es ist auch zu sehen, dass die Parameteränderung bei großen Ableitungswerte wie $\partial\theta \geq 1e100$ numerisch stabil bleibt, was aber bei SGD und Momentum SGD nicht immer der Fall ist. Das ist eine sehr wichtige Eigenschaft, weil so kein Bedarf besteht, die Lernrate immer sehr klein zu setzen oder sehr aufmerksam auf die Normalisierung der Ausgaben zu achten.

Ein alternativer Algorithmus zum RMS Prop ist der Adaptive Momentum Estimation von Kingma and Ba (2014). Wie aus Abbildung 4.12 zu sehen ist, kann er weniger sensibel gegenüber Oszillationen sein, was vom Vorteil sein kann, aber auch weniger sensibel gegenüber Monotonie, was vom Nachteil sein kann. Der Algorithmus ist abhängig von der Iteration und die beiden Terme $\hat{\mu_j} = \frac{\mu_j}{1-\beta_1^{i+1}}$ und $\hat{\sigma_j} = \frac{\sigma_j}{1-\beta_2^{i+1}}$ korrelieren damit negativ. So wird sichergestellt, dass im Verlauf des Trainings die Lernrate dynamisch verkleinert wird, um im späteren Verlauf lokal optimale Stellen nicht zu überspringen.

Ein Einblick in die Effektivität aller fünf Algorithmen ist in Abbildung 4.14 zu sehen. Bei günstiger Initialisierung konvergieren alle zu dem globalen Minimum $x = -0.2$ und bei ungünstiger konzentrieren sich alle in einem lokalen Minimum und können von da nicht auskommen. Das Verhalten ist stark von der zu optimierenden Funktion abhängig. Für andere Funktionstypen können die Resultate anders aussehen. Die Auswahl des Optimierers zählt zu den Metaparametern des Modells und ist auch von der Aufgabe selbst abhängig.

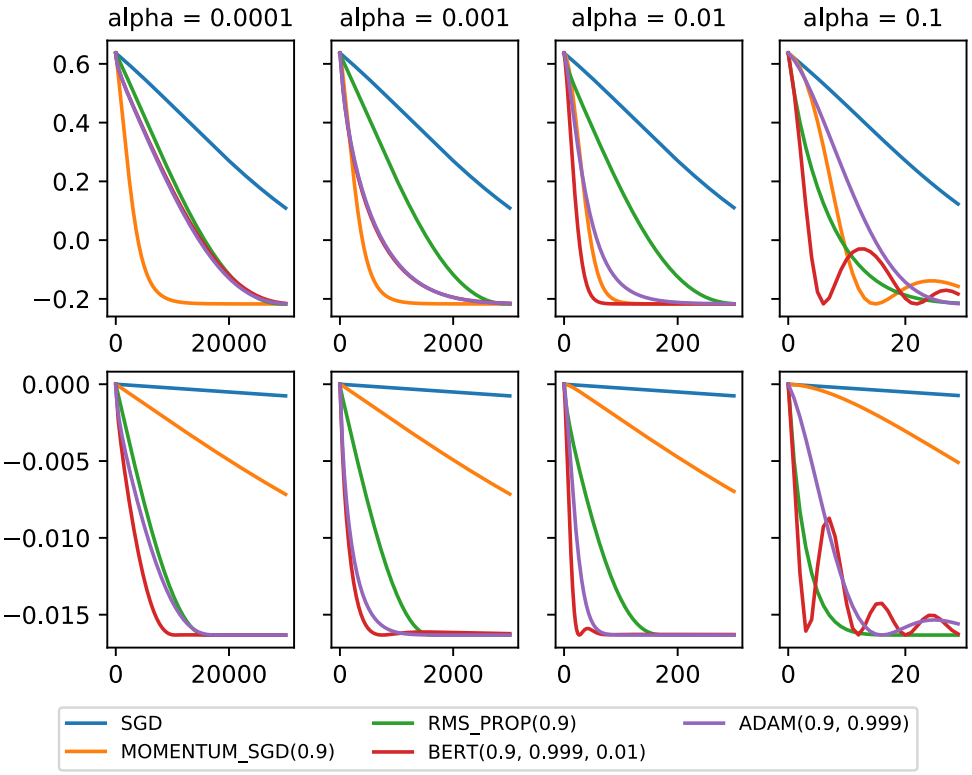

Abbildung 4.14: Minimierung von $f(x) = \frac{1}{x}\sin(x)$ mit vier verschiedenen Lernraten und zwei verschiedenen Initialisierungen. Globales Minimum ist bei $x = -0.2$

4.10 Lernrate

Die Lernrate $\alpha \in [0, 1]$ der Optimierung steuert die Geschwindigkeit der Parameteränderung. Sie korreliert positiv mit der Wahrscheinlichkeit, dass beim Optimieren die lokal optimale Stelle übersprungen wird, weil je größer sie ist, desto um größere Beträge werden die Parameter geändert. Allerdings je kleiner sie ist, desto langsamer wird das Training. Ziel ist es, ein Balance zwischen Optimalität und Trainingsdauer zu finden. Einige Optimierungsalgorithmen wie ADAM und RMS Prop sind weniger sensibel gegenüber der Lernrate und das bringt mehr Freiheit bei der Auswahl.

Die Lernrate wird im Rahmen des Trainings dynamisch oder konstant gehalten. Eine konstante Lernrate, die zu akzeptablen Resultaten führt, ist schwierig einzuschätzen. Außerdem hat eine konstante Wahl den Nachteil, dass jede Parameteränderung unabhängig vom Trainingsschritt wird. Eine alternative ist die dynamische Lernrate. Diese lässt sich in jedem Trainingsschritt ändern. Eine Vermutung ist, dass die Optimierung schneller in der Nähe der lokal optimalen Stelle kommt, wenn am Anfang die Lernrate groß ist, so dass große Schritte gemacht werden, und dann sukzessiv verkleinert wird, so dass die Parameteränderungen präziser werden. Ein Vorgehen, was diesem Ansatz folgt, ist die dynamische Lernrate mit polynomial wachsendem Zerfall. Sie interpoliert für $w_1 \in \mathbb{N}$ viele Trainingsschritte zwischen einer Anfangs- und einer Ziellernrate $\alpha_1 \in [0, 1]$ bzw. $\alpha_2 \in [0, 1]$ mit $\alpha_1 \geq \alpha_2$. Durch den Faktor $q \in \mathbb{N}$ wird die Geschwindigkeit des Zerfalls gesteuert. Für Trainingsschritt j ergibt sich folgende Lernrate α_j.

$$\textbf{Polynomial Decay}: \quad i_j = \frac{j}{w_1} \qquad \alpha_j = (\alpha_1 - \alpha_2)((1 - i_j)^q) + \alpha_2$$

Ein Nachteil dieser Technik ist, dass wenn sich das Modell am Anfang des Trainings nah an dem lokalen Optimum befindet und α_1 zu groß ist, dann werden große Parameteränderungen eine Entfernung von der lokal optimalen Stelle bringen. Das wird im BERT adressiert. Durch eine Anzahl an Trainingsschritten zum Aufwärmen wird versucht das zu korrigieren. Für $w_2 \in \mathbb{N}$ viele Schritte wird die Lernrate kontinuierlich linear schnell vergrößert. Und ab dann wird sie mit einem polynomialen Zerfall verkleinert. Für eine Anfangslernrate α_1, eine Ziellernrate α_2, w_1 viele Schritte für Interpolation, w_2 Schritte zum Aufwärmen und Faktor des polynomialen Zerfalls q ergibt sich im Trainingsschritt j die folgende Lernrate α_j.

$$\textbf{BERT Decay}: \quad i_j = \frac{j}{w_1} \quad \beta_j = (\alpha_1 - \alpha_2)((1 - i_j)^q) + \alpha_2$$
$$\alpha_j = \beta_j \quad j > w_2$$
$$\alpha_j = \frac{j}{w_2}\alpha_1 \quad j \leq w_2$$

Ein Nachteil ist, dass wenn sich das Modell nach der Aufwärmung weg von der lokal optimalen Stelle befindet, wird es noch schwieriger diese zu erreichen, da die Lernrate polynomial schnell gedämpft wird. Im BERT wird $q = 1$ und $\alpha_2 = 0$ gesetzt. Das bedeutet, dass der Zerfall linear schnell wächst und gegen 0 konvergiert. Die Anzahl an Trainingsschritte für die Interpolation wird gleich der gesamten Anzahl an Trainingsschritten gesetzt. Zum Aufwärmen wird die Hälfte aller Trainingsschritte benutzt.

Im ursprünglichen Transformer wird einem ähnlichen Ansatz gefolgt. Der Unterschied ist, dass die Lernrate nicht nur von dem Trainingsschritt, sondern auch von der versteckten Größe des Modells abhängig ist. Für versteckte Größe $H \in \mathbb{N}$ und Anzahl an Trainingsschritten zum Aufwärmen $w_2 \in \mathbb{N}$ ergibt sich im Trainingsschritt j die folgende Lernrate α_j.

$$\textbf{Transformer}: \quad \alpha_j = \frac{1}{\sqrt{H}} \min(\frac{1}{\sqrt{j}}, j\frac{1}{w_2^{1.5}})$$

Die Lernrate α_j korreliert negativ mit der versteckten Größe H, was bedeutet, dass Modellen mit vielen Parametern vorsichtiger mit den Parameteränderungen vorgehen werden. Ein Vorteil ist auch, dass keine Einschätzung einer initialen Lernrate benötigt wird. Das erspart bei Unsicherheit viel Arbeit.

Ein Einblick in die Funktionsgraphen der vier vorgestellten Algorithmen ist in folgende Abbildung zu sehen.

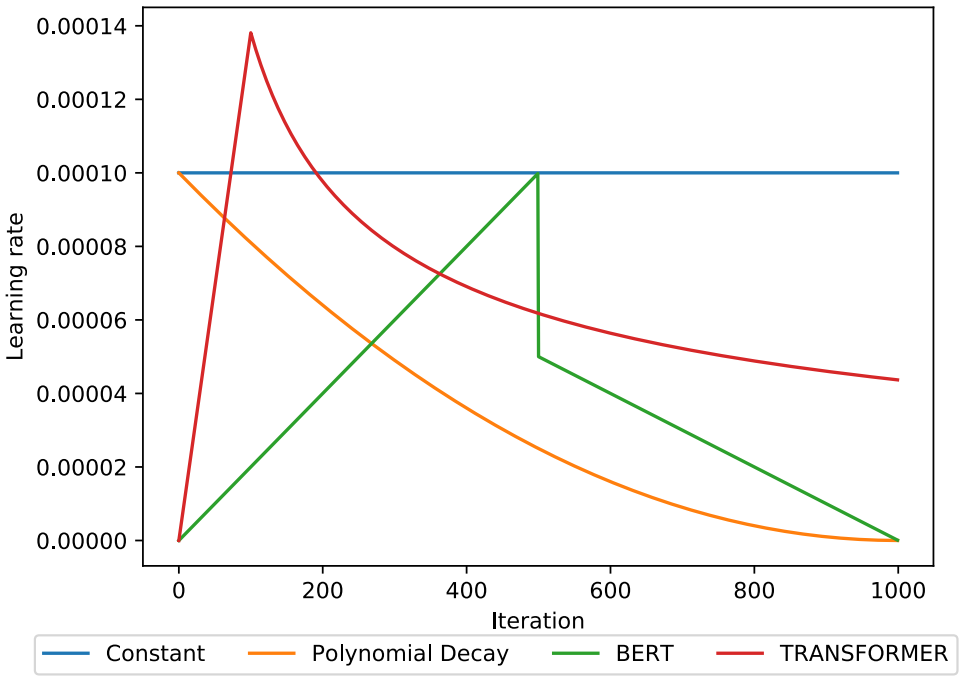

Abbildung 4.15: Ablauf der Lernrate der vier Schedulings

4.11 Zielfunktion

Die Auswahl an Zielfunktionen ist umfangreich und zählt zu dem Metaparameter des Modells. Es existieren Heuristiken und Kriterien zur Unterstützung dieser Auswahl, aber es besteht für keine Funktion a-priori Wissen, wie gut sie die Aufgabe löst, oder Garantie, dass sie die besten Resultate erreicht. Die Auswahl ist auch an die Daten selbst bedingt. Wenn Mangel an Daten und Annotation oder Ungenauigkeit und Rauschen in die Daten bestehen, ist die Auswahl beschränkt. Wenn eine auf die zu lösende Aufgabe spezialisierte Zielfunktion formuliert wird, kann sie zur Verbesserung der Genauigkeit führen, denn so wird das Training stärker auf die Aufgabe fokussiert. Das versucht BERT zu tun.

BERT löst zwei Aufgaben und verwendet eine Zielfunktion gemäß dieser Aufgaben. Die erste besteht darin zu einem Paar aus zwei Sequenzen, wo bestimmte Tokens maskiert sind, die maskierten Tokens vorherzusagen. Die zweite ist die Kohärenz der zwei Sequenzen einzuschätzen - zu entscheiden, ob die beiden aufeinanderfolgend sind - ob eine semantische Beziehung zwischen den beiden existiert. Die erste Aufgabe ist eine Multiklassen-Klassifizierung. In jedem Paar zweier Sequenzen werden höchstens e viele Tokens maskiert. Dabei ist e von der Länge des Paares abhängig und kann maximal E_{max} sein. Die Anzahl aller Tokens im Vokabular ist V. Die erste Aufgabe benutzt Ausgabe $\hat{Y}_1 \in \mathbb{R}^{e \times V}$, wo für jede maskierte Stelle $j \leq e$ der Index des größten Wertes im Vektor $\hat{Y}_{1,j} \in \mathbb{R}^V$ dem Index eines Tokens des Vokabulars entspricht. In die zweite Aufgabe wird eine binäre Klassifizierung gemacht. Dafür wird eine Ausgabe $\hat{Y}_2 \in \mathbb{R}$ benutzt, die eine Wahrscheinlichkeit dafür gibt, ob die beiden Sequenzen aufeinanderfolgend sind.

Die Zielfunktion muss stark und negativ mit der Ähnlichkeit zwischen der Ausgabe $\hat{Y}$ des Modells und der erwarteten Ausgabe Y korrelieren. So wird sie klein, wenn das Modell eine richtige Einschätzung macht, und groß, wenn die Einschätzung falsch ist. Zwei mögliche Kandidaten mit dieser Eigenschaft sind die folgenden Funktionen.

$$\textbf{Mittlerer quadratischer Abstand} : F(\hat{Y}, Y) = \sum_j (\hat{y}_j - y_j)^2$$

$$\textbf{Kreuzentropie} : H(\hat{Y}, Y) = \sum_j p(y_j) \log(\hat{y}_j)$$

Für Klassifikation ist die Kreuzentropie geeigneter als den mittleren quadratischen Abstand, denn im Fall von einer Binomial- oder Multinomialverteilung (2 oder mehr möglichen Ausgaben/Klassen) werden große Abweichungen von dem erwarteten Wert härter bestraft. MSE wird häufiger für Regression benutzt - zum Beispiel bei YOLO v3 für Objektdetektion.

Die Sensibilität der beiden Funktionen auf Abweichungen von dem erwarteten Wert ist in die folgenden Abbildung zu erkennen.

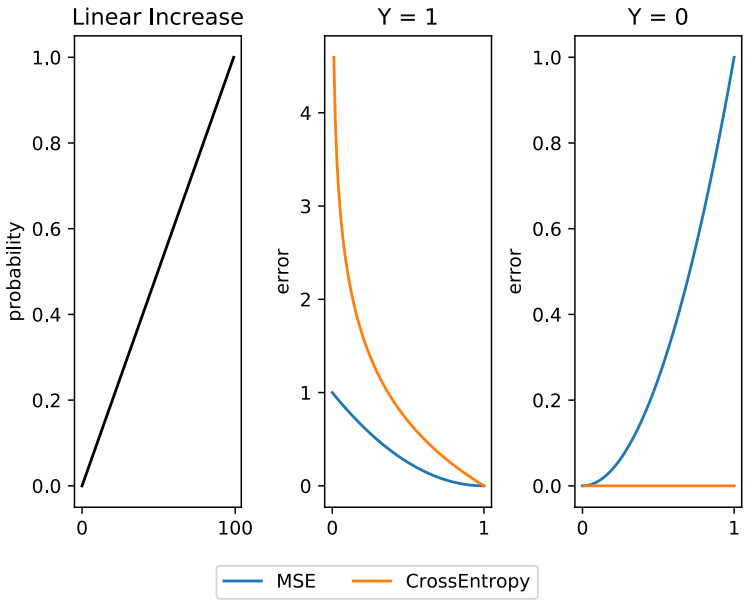

Abbildung 4.16: Auswirkung der Modellausgabe auf die Zielfunktion

Für MSE gilt $C \geq \textbf{MSE}(\hat{Y}, Y) \geq 0$ in Rahmen einer Multinomilverteilung mit $C \in \mathbb{R}$ die Anzahl an Klassen. Bei Kreuzentropie sind nur die Stellen von Bedeutung, deren erwarteten Wahrscheinlichkeiten nicht 0 sind. Dementsprechend wird nur an diesen Stellen optimiert. Das kann vom Nachteil sein, da auch die 0-Stellen richtig klassifiziert werden müssen. Um das sicherzustellen, muss das Training aus jeder Klasse genug Trainingsbeispiele haben, weil nur so das Trainings bzgl. jeder Klasse gleichmäßig und hinreichend durchgeführt wird. Das bedeutet, dass das Vokabular eine gute Repräsentativität aufweisen muss, denn andernfalls werden Klassen von Tokens, die selten oder nicht vorkommen, selten oder nicht optimiert. Es ergibt sich das folgende Maximierungsproblem für die Auswahl des Vokabulars $\hat{V}$.

$$arg\max_{\hat{V}} = \frac{\text{Häufigkeit aller Tokens in } \hat{V}}{\|\hat{V}\|} = \frac{\sum_{s \in \hat{V}} \|s\|}{\|\hat{V}\|} \qquad \|\hat{V}\| = W$$

Mit $\|s\|$ wird die Häufigkeit des Vorkommens des Tokens s und mit $\|\hat{V}\|$ die Anzahl an Tokens in das Vokabular $\hat{V}$ bezeichnet. Eine kleine Anzahl an Tokens hat den Nachteil, dass die Sequenzen eine hohe Unvollständigkeit aufweisen werden. Dementsprechend wird auch nach einer bestimmten Anzahl an Tokens W gesucht. Die Maximierung kann gelöst werden, indem aus den sortierten akkumulierten Häufigkeiten die ersten W Tokens selektiert werden..

Kapitel 5

Datenvorbereitung

Die Auswahl der Datenrepräsentation ist an die Eigenschaften der vorhandenen Daten bedingt. Mathematische Formeln müssen so repräsentiert, dass keine Eigenschaften wie Informationsinhalt oder Ordnung verloren gehen und gleichzeitig tauglich als Eingabe für BERT bleiben. Zwei solche Repräsentation werden in diesem Kapitel vorgestellt. In Rahmen dieser Masterarbeit werden mathematische Formeln aus arXiv in Form von XML-Bäumen als Trainingsdaten verwendet. Da sich diese stark von den natürlichsprachlichen Sätzen unterscheiden, ist eine Modifikation an dem Datenvorbereitungsansatz des ursprünglichen NLP-BERT notwendig. NLP-BERT sagt Kohärenz und maskierte Tokens vorher. MATH-BERT tut das auch. Dafür braucht es aber eine Definition der Kohärenz bzgl. mathematischer Formeln und ein Vorgehen zur Auswahl der Trainingsdaten. Diese werden im Unterkapitel Kohärenz erklärt. Wie jedes Paar zweier Formeln zu einer Sequenz fusioniert wird, ist im Unterkapitel Fusion zu sehen. Wie die Maskierung von Tokens in die entstandenen fusionierten Sequenzen funktioniert, wird im Unterkapitel Maskierung vorgestellt. Zwei mögliche Datenvorbereitungsansätze für die Fine-Tuning-Phase werden im Unterkapitel Fine-Tuning Datenformate präsentiert.

5.1 Datenlayout

Der arXiv-Datensatz beinhaltet Artikel aus Bereiche wie Physik, Mathematik, Informatik, Statistik, Finanzmathematik und Biologie. Diese sind im LaTeX-Format vorhanden. Mithilfe eines an der TU-Dortmund entwickelten Tools können die einzelnen Abschnitte der Artikel und die dazu gehörigen Formeln extrahiert und in MathML-Format überführt werden. Der somit entstandene Datensatz aus Artikeln beinhaltet den Zusammenhang zwischen Artikel und ihren Formeln. Das macht ihn geeignet für Semi-Supervised Learning und wird in Rahmen dieser Masterarbeit für das Pre-Training von BERT verwendet.

5.2 Vokabular und Sequenzlänge

BERT benötigt eine konstante Sequenzlänge und ein Vokabular. Um diese zu ermitteln, sind 1.348.373 Artikel aus arXiv als Datensatz verwendet und daraus nur diese extrahiert, die gültige Formeln beinhalten. So haben sich 870.654 Artikel ergeben und diese wurden im Verhältnis 80 zu 20 in Trainings- und Testdaten (696.523 zu 174131) aufgeteilt. Im Anschluss ist das Vokabular nur anhand der Trainingsdaten generiert worden. Die Trainingsdaten haben mehr als 27.800.000 Formeln und bestehen aus mehr als 600.000 einzigartigen Elementknoten. Zur Repräsentation von 98.8% der akkumulierten Häufigkeiten sind nur 512 Token (0.1 % aller Tokens) notwendig. Das bedeutet, dass im Vergleich zu der Repräsentation vieler NLP-Aufgaben (zum Beispiel GPT-2 zum Übersetzen von Sequenzen mit Vokabular mit 50257 Wörtern [1]) ist die Repräsentation mathematischer Formeln mit einer deutlich kleinerer Vokabulargröße möglich. Außerdem sind 95.0% aller Formeln mit höchstens 256 Tokens präsentierbar, was mehr als 450 Mal kürzer ist als die längste im Datensatz vorhandene Sequenz und 2 Mal kürzer als diese vom Google BERT.

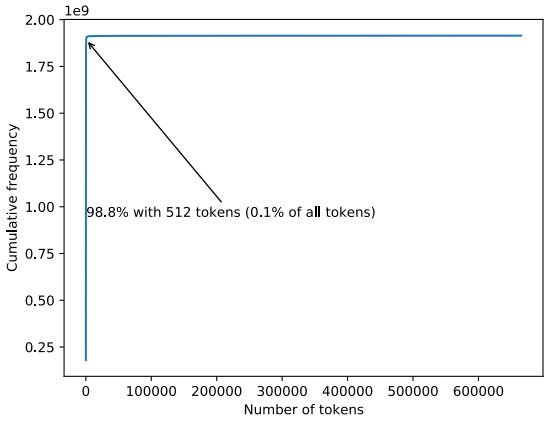

Abbildung 5.1: Akkumulierte Häufigkeiten der vorkommenden Tokens

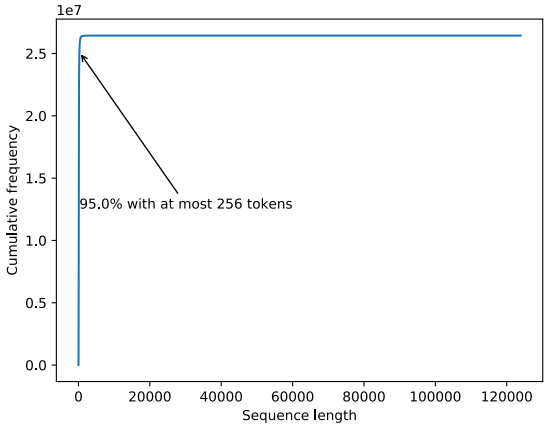

Abbildung 5.2: Akkumulierte Häufigkeiten der Sequenzlängen

Wie aus Abbildung 5.2 zu sehen ist, wäre eine Auswahl von 512 oder 1024 als Sequenzlänge repräsentativer, da so kommt die kumulative Häufigkeit näher an 100%. Allerdings führt eine Verdoppelung der Sequenzlänge zum Quadrieren der benötigten Speicherressourcen in Multi-Head-Attention, da jedes Paar von Tokens berücksichtigt wird und insgesamt quadratisch viele sind. Um die Batchgröße auf mindestens 64 zu setzen, ist als Sequenzlänge 256 ausgewählt. So werden insgesamt 26.439.075 Formeln abgedeckt und insgesamt 105.859.321 Paare daraus gebildet.

[1] https://huggingface.co/transformers/model_doc/gpt2.html

5.3 Repräsentation mathematischer Ausdrücke

Die visuelle Repräsentation mathematischer Formeln liefert keine Information über die Reihenfolge der Anwendung der mathematischen Operatoren und Funktionen. Zum Beispiel bei dem Ausdruck $1 + 5 * 4$ kann man ohne Vorwissen und nur anhand der visuellen Repräsentation die Reihenfolge der Anwendung der mathematischen Operatoren nicht erkennen. Es ist nicht eindeutig, ob die Multiplikation oder die Addition eine höhere Priorität hat. Der Mensch und viele Interpreter kennen die Reihenfolge und alles, was visuell fehlt, aber ein Modell muss das lernen. Eine möglichst eindeutige Repräsentation lässt weniger Raum für fehlerhafte Interpretationen und kann nur vom Vorteil beim Training sein. Eine domänenspezifische Sprache, die das adressiert ist die MathML. Sie hat um die 140 Element- und 12 Attributknoten [2] und dadurch lässt sich jeder gültiger mathematischer Ausdruck beschreiben. Es gibt verschiedene Algorithmen zur Transformation von mathematischen Formeln aus der Graph- in eine Sequenz-Form. Diese Algorithmen unterscheiden sich vor allem in die Reihenfolge der Baumiteration und das Layout der Knotenspeicherung.

Eine erste Möglichkeit für diese Transformation ist indem über die Baumstruktur in einer Präfix-Reihenfolge iteriert wird und die dabei besuchten Knoten in die Reihenfolge der Iteration gespeichert werden - so wie die Datenvorbereitung von Lample and Charton (2019) zum Lösen von Differentialgleichungen und symbolischer Integration. Die Knoten des Baumes werden zu Elementen der Sequenz. Für die entstandene Sequenz ist eine zusätzliche Transformation benötigt und vor allem eine Quantifizierung damit sie die Kompatibilität als Eingabe für ein numerisches Model erreicht. Dies kann gelöst werden indem aus einem vortrainierten Vokabular jedem Element der Sequenz eine ganze Zahl zugeordnet wird. Um die Beziehungen zwischen Knoten und Kinderknoten stärker zu betonen, wird die ausgewählte Zahl (das Token) vor und nach den inneren Knoten positioniert so, dass sowohl der Inhalt als auch die Ordnung der Formeln komplett behalten bleiben. Aus dieser Sequenz ist der Baum vollständig rekonstruierbar. Ein Beispiel dieser Technik ist in die folgenden Abbildungen zu sehen. Mit MathML[3] wird der Ausdruck $1 - (2/a)^{-1/4n}$ zuerst in einer XML-Struktur überführt - Abbildung 5.3. Dann mithilfe einer Iteration in Präfix-Reihenfolge wird eine Sequenz generiert. Jedem Element wird anschließend aus dem Vokabular in Abbildung 5.4 ein Token zugeordnet so, dass die Sequenz in Abbildung 5.5 entsteht.

[2] https://reference.wolfram.com/language/XML/tutorial/MathML.html
[3] https://www.w3.org/TR/MathML3/

```
<mrow>
  <mn>1</mn>
  <mo>-</mo>
  <msup>
    <mrow>
      <mo fence="true">(</mo>
      <mfrac>
        <mn>2</mn>
        <mi>a</mi>
      </mfrac>
      <mo fence="true">)</mo>
    </mrow>
    <mrow>
      <mo>-</mo>
      <mfrac>
        <mn>1</mn>
        <mrow>
          <mn>4</mn>
          <mi>n</mi>
        </mrow>
      </mfrac>
    </mrow>
  </msup>
</mrow>
```

Abbildung 5.3: $1-(2/a)^{-1/4n}$ als MathML XML-Struktur

```
<mrow> = 0
<mn>1</mn> = 1
<mo>-</mn> = 2
<msup> = 3
<mo fence="true">(</mo> = 4
<mfrac> = 5
<mn>2</mn>= 6
<mi>a</mi> = 7
<mo fence="true">)</mo> = 8
<mn>4</mn> = 9
<mi>n</mi> = 10

.
```

Abbildung 5.4: Vokabular

```
0
  1
  2
  3
    0
      4
      5
        6
        7
      5
      8
    0
    0
      2
      5
        1
        0
          9
          10
        0
      5
    0
  3
0
```

Abbildung 5.5: Sequenz

Die Daten von Lample and Charton (2019) haben den Vorteil, dass Operatoren und Funktionen immer vor den Argumenten positioniert sind. So können Klammern bei der Kodierung weggelassen werden. Bei MathML können sich diese auf die gleiche Stufe befinden wie die Argumente. Dementsprechend werden Metasymbole wie Klammern auch kodiert. Bei diesem Ansatz kann die Positionierung von umschließenden Elementknoten vor und nach dem Argument erspart werden, indem nur vor den inneren Knoten eine Positionierung stattfindet, denn so ist die Baumstruktur auch wiederherzustellen. Das führt im besten Fall zu einer Verkürzung der Sequenzlänge um Faktor zwei, aber kann zu einer Reduzierung der Genauigkeit des Modells führen, weil es von selbst die Unterscheidung zwischen öffnenden und schließenden Elementknoten erlernen muss. In Rahmen dieser Masterarbeit werden für öffnenden und schließenden Elementen die gleichen Kodierungen benutzt. Das führt dazu, dass die Größe des Vokabulars verkürzt wird, was sich auf die Größe des Modells auswirkt.

Die Graph basierten Ansätze zur Formelrepräsentation sind weitverbreitet und ein solcher wird auch von Luo and Liu (2018) zur automatischen Berechnung von Ab-

leitungen eingesetzt. Allerdings im Vergleich zu dem oben vorgestellten Ansatz werden die Bäume mit einer schichtweise iterierenden Technik kodiert. Bei dieser wird erstmal die erste Schicht, dann die zweite und s.w. kodiert. Innerhalb jeder Schicht werden die Tokens von links nach rechts durchgelaufen und jeweils der Knoten und seine Kinderknoten kodiert. Ein Beispiel der Kodierung ist in die folgende Abbildung zu sehen.

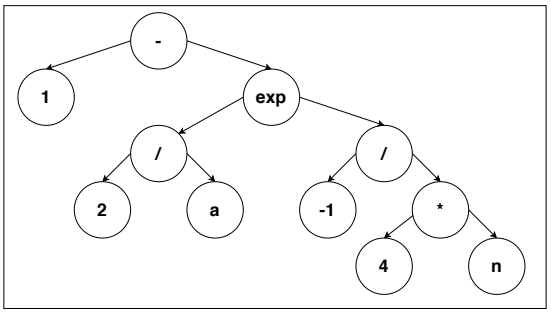

Abbildung 5.6: $1 - (2/a)^{-1/4n}$ als Baum

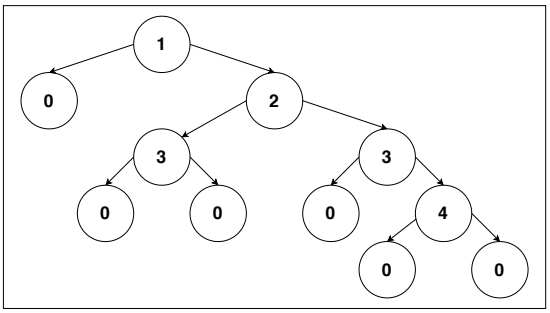

Abbildung 5.7: Kodierter Ausdruck

Es wird jedem Knoten, der eine mathematische Funktion oder Operator ist, eine Zahl $e \geq 1$, und jedem Knoten, der eine Zahl oder Variable ist, die Zahl 0 zugeordnet. Anschließend wird schichtweise über den Baum und in jeder Schicht entlang der Knoten von links nach rechts iteriert. Für jeden Knoten werden seine Kodierung und diese seiner Kinder am Ende der Sequenz eingefügt. Es ergibt sich die folgende Kodierung für das Beispiel.

$$1,0,2 \quad 2,3,3 \quad 3,0,0,3,0,4 \quad 4,0,0$$

Der Vorteil dieses Ansatzes ist, dass er sich stärker auf die Beziehung zwischen Knoten und Kinderknoten fokussiert. Außerdem korreliert die Position des Knotens im Sequenz stark und positiv mit seiner Schicht in Baum. Allerdings jeder Variable und Zahl wird die gleiche Zahl zugeordnet, was dazu führt, dass der Baum anhand der Sequenz nicht zurück konstruierbar und dass das Modell keine Unterscheidung von Variablen und Zahlen erlernen kann. Dies kann aber korrigiert werden, indem für diese Elementknoten auch eindeutige Zahlen ausgewählt werden.

5.4 Kohärenz

NLP-BERT sagt das Aufeinanderfolgen von Sätzen vorher. Die natürlichen Sprachen haben den Vorteil, dass aufeinanderfolgende Sätze eine semantische Beziehung haben, denn ein Satz kann den davorstehenden Satz ergänzen. Außerdem setzen die zu einem Satz folgenden Sätze häufig die Beschreibung fort. Bei mathematischen Formeln ist das nicht unbedingt der Fall, denn zwei aufeinanderfolgende Formeln können

zwei komplett unterschiedliche Konzepte beschreiben. Außerdem nur weil in einem Artikel sie folgend sind, ist keinen Grund zur Annahme, dass das im Allgemein der Fall ist. Wenn natürlichsprachliche Sätze wissenschaftliche Konzepte aus vertrauenswürdigen Quellen beschreiben - so wie die Bücher das machen - dann ist die Beschreibung im Allgemein gültig. Dementsprechend, wenn eine Konstellation aus Sätzen einmal vorkommt, kann angenommen werden, dass diese hinreichend repräsentativ ist. Bei Formeln ist das nicht unbedingt so, denn für mathematische Formeln desselben Artikels ist die Relation, ob diese aufeinanderfolgend sind, nicht plausible und ausdrucksstark genug, um eine Aussage über die Formeln oder den Artikel zu treffen. In Rahmen dieser Masterarbeit wird eine Alternative vorgeschlagen - die Relation, ob zwei Formeln aus demselben Artikel stammen. Man kann annehmen, dass diese Relation für das Erschaffen eines grundlegenden Verständnisses für Mathematik entscheidend ist, weil unterschiedliche Artikel, die gleiche Konzepte beschreiben oder semantisch ähnlich sind, können dieselbe Formeln haben. Für zwei Sequenzen x und y ergeben sich die folgenden Relationen $\sim_1$ und $\sim_2$.

NLP-BERT : $x \sim_1 y \Leftrightarrow$ y folgt x

MATH-BERT : $x \sim_2 y \Leftrightarrow$ x und y sind aus gleichem Artikel

Im Kapitel Zielfunktion ist erklärt, dass für eine Optimierung mit Kreuzentropie aus jeder Klasse genug Trainingsdaten benötigt werden, weil nicht nur die 1-Stellen berücksichtigt werden müssen. Dementsprechend sollen negative Trainingsbeispiele gebildet werden. Zum Pre-Training von NLP-BERT sind Textartikel aus Wikipedia und Bücher aus BooksCorpus verwendet. Diese sind in Abschnitte gruppiert und jeder Abschnitt A_j hat W_j viele Sätze. Für jeden Satz i eines Abschnittes j werden anhand einer $\sim \mathcal{B}(q, \alpha)$ Binomialverteilung q-viele unabhängige Zufallsexperimente durchgeführt. Bei jedem davon wird mit Wahrscheinlichkeit α den nächsten Satz $j+1$ aus dem Abschnitt oder mit Wahrscheinlichkeit $1-\alpha$ einen beliebigen anderen Satz i aus einem beliebigen anderen Abschnitt ausgewählt. Der Parameter α wird auf 0.5 gesetzt, so dass es gleich wahrscheinlich ist positive und negative Beispiele zu bilden. Der Parameter q sorgt dafür, dass zu jedem Satz mehr als nur ein Trainingsbeispiel gebildet wird. So formiert sich mehr Variation in die Trainingsdaten.

Die Trainingsdaten für MATH-BERT bestehen aus W vielen Artikeln. Jeder Artikel j hat W_j viele Formeln. Die Formel $F_{j,i}$ ist die i-te Formel des j-ten Artikel. Für jede Formel $F_{j,i}$ werden anhand einer $\sim \mathcal{H}(q, \alpha)$ hypergeometrische Verteilung q-viele unabhängige Experimente durchgeführt und bei jedem mit Wahrscheinlichkeit α aus dem gleichen Artikel eine beliebige Formel $F_{q,i}$ oder mit Wahrscheinlichkeit $1-\alpha$ aus einem beliebigen anderen Artikel eine beliebige Formel $F_{x,y}$ ausgewählt. Diese Technik hat den Vorteil, dass jede Formel $F_{j,i}$ maximal W_i viele positive Trainingsbeispiele haben kann. Wenn aus jedem Artikel jedes Paar zweier Formeln in die Trainingsdaten

kommt, dann entstehen in jede zwei Artikel A_i und A_j jeweils vollständige Graphen bzgl. der Relation $\sim_2$. So wird sichergestellt, dass Ähnlichkeiten der Art $F_{x,i} = F_{y,j}$ und $F_{w,i} = F_{z,j}$ durch die Relationen $F_{x,i} \sim_2 F_{w,i}$ und $F_{y,j} \sim_2 F_{z,j}$ in die Trainingsdaten involviert werden.

Die Annahme ist, dass ähnliche Artikel ähnliche Formeln haben. Dementsprechend ist es wichtig, dass genau die Relationen dieser ähnlichen Formeln in die Trainingsdaten vorkommen. Wenn das nicht der Fall ist, dann wird es schwieriger einen Anknüpfungspunkt ähnlicher Artikel zu finden. Ein Beispiel einer günstiger und ungünstiger Auswahl der Relation $\sim_2$ ist in die folgenden zwei Abbildungen zu sehen. In die linke Abbildung ist die Relation $F0 \sim_2 F1$ weder aus Artikel $A1$ noch aus $A2$ extrahiert. Diese Relation kann entscheidend für die Annahme sein, dass $A1$ und $A2$ ähnliche Artikel sind. In die rechte Abbildung ist die Relation $F0 \sim_2 F1$ involviert und deshalb kann man annehmen, dass diese Auswahl günstiger ist.

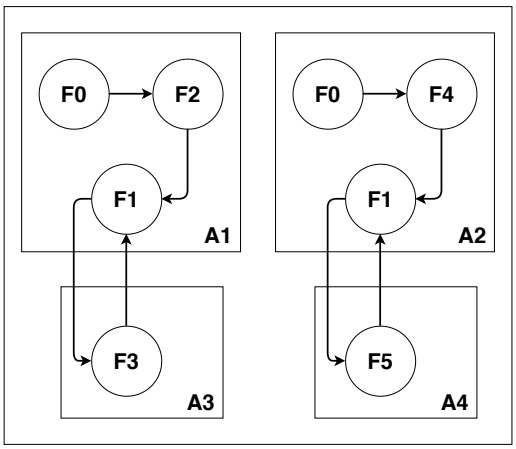

Abbildung 5.8: Ungünstige Auswahl

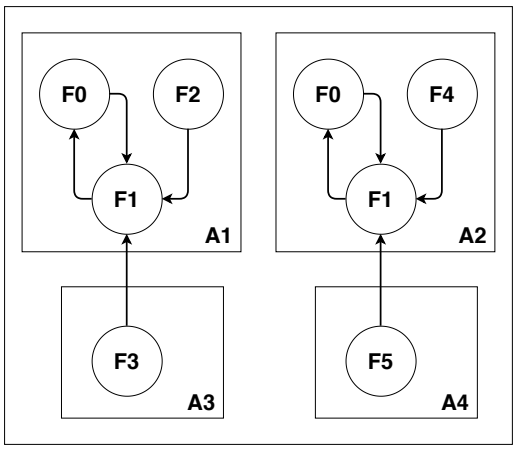

Abbildung 5.9: Günstige Auswahl

Um sicherzustellen, dass alle entscheidenden Relationen involviert werden, soll der vollständige Graph bzgl. der Relation $\sim_2$ gebildet werden. Das hat allerdings den Nachteil, dass die Anzahl an Trainingsdaten deutlich größer wird, denn für Artikel A_j mit W_j Formeln, werden W_j^2 viele Relationen entstehen. Die Anzahl an Trainingsdaten wächst quadratisch bzgl. q der Binomialverteilung $\mathcal{B}(q, \alpha)$. Die Trainingsdauer wird entsprechend quadratisch länger sein. Der Trade-Off ist zwischen Qualität der Daten und Trainingsdauer. Man soll eine gute Einschätzung für den Parameter q finden. Möglichkeiten sind zum Beispiel den Mittelwert an Formeln pro Artikel, einen Perzentil der akkumulierten Häufigkeiten oder einfach ein Vielfaches der gewünschten Anzahl an Trainingsdaten zu nehmen. Im BERT wird ein Faktor von 5 genommen. In MATH-BERT wird das auch so gemacht.

5.5 Fusion

Jedes Paar zweier Formeln soll zu einer Sequenz fusioniert. Dafür wird das Vokabular $\hat{V}$ wie folgt erweitert:

$$\hat{V} = \hat{V} \cup \{\text{[CLS]}, \text{[SEP]}\}$$

$$\text{[CLS]} = \text{Anfang des Paares}$$

$$\text{[SEP]} = \text{Trennsymbol zwischen den beiden Sequenzen und am Ende des Paares}$$

Das [CLS] Token wird vor jedem Paar zweier Formeln $F_{i,x}$ und $F_{j,y}$ positioniert. Es steht für Classification und wird für das Vorhersagen der Kohärenz verwendet. Das [SEP] Token wird zwischen den beiden Formeln und am Ende des Paares positioniert und steht für Separation - Trennung der beiden Formeln. Für zwei beliebige Formeln $F_{i,x}$ und $F_{j,y}$ ergibt sich die folgende Sequenz Y.

$$Y = \text{[CLS]}\ \ F_{i,x}\ \ \text{[SEP]}\ \ F_{j,y}\ \ \text{[SEP]}$$

Das [SEP] Token markiert die Grenzen der beiden Sequenzen in Y. Allerdings für ein beliebiges $y \in Y$ reicht das nicht, um nur anhand von y zu erkennbar, ob y aus $F_{i,x}$ oder aus $F_{j,y}$ stammt. Dementsprechend wird in Rahmen der Datenvorbereitung eine Sequenz zur Segmentierung $\hat{Y}_{\text{seg}}$ gebildet, die die Zugehörigkeit der Tokens zu den Formeln wiedergibt. Y_{seg} hat die gleiche Länge und Ordnung wie Y. Für jedes $y_{\text{seg}} \in Y_{\text{seg}}$ mit Index q gilt $y_{\text{seg}} = 1$, wenn die Stelle Y_q aus der zweiten Formel $F_{j,y}$ stammt.

$$Y_{\text{seg},j} = 1 \Leftrightarrow Y_j \in F_{j,y} \qquad Y_{\text{seg},j} = 0 \Leftrightarrow Y_j \in F_{i,x}$$

Anhand von Y_{seg} werden in Rahmen der Einbettung die Tokens der zweiten Sequenz mithilfe einer Gewichtsmatrix stärker betont, so dass für das Modell eine Orientierung besteht, die Zugehörigkeit der Tokens zu den Sequenzen anhand der Kodierung zu erkennen. Das ist entscheidend, um die Kohärenz vorherzusagen, denn das Modell kann diese entscheiden, nur wenn erstmals die Tokens der beiden Formeln erkannt sind.

BERT hat eine Eingabesequenzlänge von Y_{MAX}. Allerdings nicht jedes Paar zweier Formeln ist exakt so lang. Wenn ein Paar kürzer ist, d.h. Y hat eine Länge $\|Y\| < Y_{\text{MAX}}$, dann werden die letzten $Y_{\text{MAX}} - \|Y\|$ Stellen mit 0 gefüllt, weil diese irrelevant für das Paar sind. Durch die Matrixmultiplikation in Dense- und Multi-Head-Attention-Schichten werden diese Stellen in Rahmen des Trainings so transformiert, dass sie keine Nullen mehr sein werden. Dementsprechend sollen diese gedämpft werden. Dazu wird in Rahmen der Datenvorbereitung eine Sequenz Y_{input} für die Eingabe gebildet. Diese hat die gleiche Länge und Ordnung wie Y. Sie kodiert für jede Stelle $j \in Y_{\text{MAX}}$, ob j Teil des Paares ist. Es gilt das folgende:

$$Y_{\text{input},j} = 1 \Leftrightarrow j \leq \|Y\| \qquad Y_{\text{input},j} = 0 \Leftrightarrow j > \|Y\|$$

5.6 Maskierung

In jedem Paar zweier Formeln werden Tokens maskiert. Das wird nach dem gleichen Prinzip wie bei NLP-BERT gemacht. Dazu wird das Vokabular $\hat{V}$ um die folgenden zwei Tokens ergänzt.

$$\hat{V} = \hat{V} \cup \{\text{[MASK]}, \text{[UNK]}\}$$
$$\text{[MASK]} = \textbf{Maskierte Stelle}$$
$$\text{[UNK]} = \textbf{Unbekanntes Token (nicht vorhanden im Vokabular)}$$

Ähnlich dem Konzept von Dropout-Schichten, wo $\alpha\%$ Stellen der Ausgabe ausgeschaltet werden, um das Modell auf unvollständige Daten besser vorzubereiten, werden bei NLP-BERT Tokens in die Trainingsdaten maskiert, so dass das Modell diese vorhersagen kann. So erlernt es die Unvollständigkeit zu korrigieren. Das bedeutet, dass das Modell ein tieferes Verständnis für Sprache erreicht, als wenn er nur die Kohärenz einschätzt. Lücken in Sätzen zu ergänzen heißt, dass das Modell die Beziehungen zwischen den Tokens erlernt hat und den grundlegenden Sinn auch bei Unvollständigkeit erkennt - so wie die Menschen das tun.

In MATH-BERT wird auch Maskierung vorgenommen. Wenn das Modell fehlende Operatoren, Zahlen oder Variablen in die Formeln korrekt vorherzusagen kann, dann bedeutet das, dass es mathematische Formeln korrigieren kann und ein grundlegendes Verständnis für die Mathematik erlernt hat. Die Annahme ist, dass das vom Vorteil für die Kohärenz und die Fine-Tuning-Phase ist.

Die Maskierung nach NLP-BERT wird durch die Parameter β und E_{max} gesteuert. Der erste steuert wie viel Prozent der Tokens maskiert werden und der zweite die maximale Anzahl an maskierten Tokens. Beide beziehen sich auf die Gesamtlänge des Paares zweier Sequenzen. Seien $F_{i,x} \in \mathbb{R}^{E_1}$ und $F_{j,y} \in \mathbb{R}^{E_2}$ zwei Formeln als numerische Sequenzen mit Anzahl an Tokens $E_1 = |F_{i,x}|$ und $E_2 = |F_{j,y}|$. Sei $Y \in \mathbb{R}^{E_1+E_2+3}$ die durch die Fusion entstandene Sequenz. Im Rahmen der Maskierung von Y werden insgesamt $e = \min(\beta(E_1 + E_2), E_{\text{max}})$ viele Indizes I von Y gemäß einer hypergeometrische Verteilung ausgewählt. Jeder Index darf höchstens einmalig ausgewählt werden. Die Indizes der speziellen Tokens [CLS] und [SEP] sind bei der Auswahl ausgeschlossen. An jeder Stelle $i \in I$ und Kodierung $w = Y_i$ wird dann gemäß einer Binomialverteilung $\sim \mathcal{B}(q = 1, \alpha = 0.8)$ entschieden, ob w durch die Kodierung des Tokens [MASK] ersetzt wird. Beim Misserfolg wird w gemäß einer Binomialverteilung $\sim \mathcal{B}(q = 1, \alpha = 0.5)$ durch sich selbst oder durch die Kodierung eines beliebigen anderen Tokens aus dem Vokabular $\hat{V}$ ersetzt. Ein Beispiel sieht wie folgt aus:

$$F_{i,x} = \{0, 1\} \quad F_{j,y} = \{2, 3, 4\} \quad \beta = 0.20 \quad E_{\text{max}} = 2 \quad I = \{2\}$$
$$\hat{Y} = \{\text{[CLS]}, 0, \text{[MASK]}, \text{[SEP]}, 2, 3, 4, \text{[SEP]}\}$$

Der Encoder im BERT kodiert Y in die Ausgabe $\hat{Y}_{\text{ENCODER}} \in \mathbb{R}^{S \times H}$. Die Ordnung der Sequenz bleibt unverändert. Das bedeutet, dass $\hat{Y}_{\text{ENCODER},j}$ die Kodierung des j-ten Tokens von Y ist.

Die Klassifikationsschicht im BERT schätzt dann insgesamt E_{max} viele Tokens in Form einer Wahrscheinlichkeitsverteilung $\hat{Y}_{\text{MASK}} \in \mathbb{R}^{E_{\text{max}} \times V}$. Das erfolgt indem aus der Encoder-Ausgabe $\hat{Y}_{\text{ENCODER}}$ die Indizes $Y_{\text{position}} \in \mathbb{R}^{E_{\text{max}}}$ selektiert, anhand der Worteinbettungsmatrix auf Dimension V projiziert und mit Softmax auf $[0, 1]$ abgebildet werden.

Es kann aber sein, dass weniger als E_{max} Tokens benötigt werden, wenn $e = \min(\beta(E_1 + E_2), E_{\text{max}}) < E_{\text{max}}$ gilt. In diesem Fall sollen nicht alle der vorhergesagten Tokens aus $\hat{Y}_{\text{MASK}}$ berücksichtigt werden, sondern nur diese, die zu Y gehören, also alle Stellen $\hat{Y}_{\text{MASK},j}$ mit $j \leq e$. Dafür werden zwei weitere Sequenzen in Rahmen der Datenvorbereitung gebildet. Die Sequenz $Y_{\text{position}} \in \mathbb{R}^{E_{\text{max}}}$ mit den Indizes der maskierten Stellen und die Sequenz $Y_{\text{tokens}} \in \mathbb{R}^{E_{\text{max}}}$ mit den maskierten Tokens. Es gilt das folgende:

$$Y_{\text{position},j} = 0 \Leftrightarrow j > e \qquad 0 < Y_{\text{position},j} < E_1 + E_2 \Leftrightarrow j \leq e$$
$$Y_{\text{tokens},j} = 0 \Leftrightarrow j > e \qquad Y_{\text{tokens},j} \in F_{i,x} \cup F_{j,y} \Leftrightarrow j \leq e$$

Um den Fokus auf die maskierten Stellen weiter zu verstärken, wird für jedes Paar Y ein Gewichtsvektor $Y_{\text{weight}} \in \mathbb{R}^{E_{\text{max}}}$ gebildet. Er ist wie folgt definiert:

$$Y_{\text{weight},j} = 0 \Leftrightarrow j > e \qquad Y_{\text{weight},j} = 1 \Leftrightarrow j \leq e$$

Dieser Vektor wird bei der Normierung der Kreuzentropie verwendet, indem die nicht aufsummierte Kreuzentropie $H(\hat{Y}_{\text{MASK}}, Y_{\text{MASK}}) \in \mathbb{R}^{E_{\text{max}}}$ damit multipliziert wird. So werden bei der Aufsummierung nur die maskierten Stellen involviert. Es ergibt sich die folgende modifizierte Kreuzentropie.

$$\begin{aligned} H(\hat{Y}_{\text{MASK}}, Y_{\text{MASK}}) &= \frac{\sum_j (Y_{\text{MASK},j} * \log(\hat{Y}_{\text{MASK},j}))Y_{\text{weight},j}}{\sum_j Y_{\text{weight},j} + \epsilon} \\ &= \text{der durchschnittliche Fehler pro maskierte Stelle} \end{aligned}$$

$\hat{Y}_{\text{MASK},j}$ ist dabei one-hot kodiert. In dem Zähler werden die maskierten Stellen stärker betont und alle Stellen mit $j > e$ gedämpft. Durch die Division wird der durchschnittliche Fehler pro maskierte Token berechnet.

5.7 Fine-Tuning Datenformate

Die Datenformate für die Fine-Tuning-Phase sind stark an die Art der zu lösenden Aufgabe gebunden. In Rahmen dieser Masterarbeit liegt der Fokus auf diskriminative und generative Arten von Aufgaben. Die erste Art trifft eine binäre Entscheidung bezüglich der Eingabe - so wie BERT entscheidet, ob zwei Formeln aus demselben Artikel stammen. Die zweite Art von Aufgaben generieren anhand der Eingabe eine Lösung - so wie BERT die maskierten Tokens vorhersagt und so wie Lample and Charton (2019) eine Lösung für eine eingegebene Differentialgleichung generieren.

Art	Vokabulargröße	Ausgabelänge
Diskriminativ	1 und 0	1
Generativ	V	beliebig

Tabelle 5.1: Arten von Aufgaben

Da die diskriminative Art sehr stark der Klassifikationsaufgabe von BERT ähneln wird für sie dasselbe Datenformat wie beim Pre-Training verwendet. Das bedeutet, dass zwei mathematische Ausdrücke generiert werden. Der erste beinhaltet das Polynom und der zweite eine Ableitung eines Polynoms. Die Ableitung kann auch falsch sein. Das wird absichtlich gemacht, so dass das Training auch negative Beispiele sieht. Die beiden Ausdrücke werden dann fusioniert und maskiert nach dem gleichen Prinzip wie beim Pre-Training. Die Klassifikationsschicht löst dann nur eine Aufgabe. Sie entscheidet, ob die Eingabesequenz zwei mathematische Ausdrücke beinhaltet, wo der erste ein Polynom und der zweite die dazugehörige Ableitung ist.

Für die generative Art besteht die Eingabe aus nur eine Sequenz. Das bedeutet, dass nichts fusioniert wird, Y_{input} nur die Eingabesequenz betrachtet und Y_{seg} keine Unterscheidung zwischen erster und zweiter Formel kodiert. Beim Pre-Training wird eine Sequenz aus maskierten Tokens geschätzt. Beim Fine-Tuning wird das auch gemacht, aber hier kodiert sie die Ableitung des Eingabepolynoms. Ziel der Optimierung ist, dass das Modell die richtige Ableitung in Form eines gültigen mathematischen Ausdrucks zu generieren erlernt.

Beim generativen Ableiten kann die Ausgabelänge länger als die Eingabelänge sein. Beim Ableiten von positiven Polynomen besteht aber das A-priori Wissen, dass die Ausgabe- und Eingabelänge der Kodierung gleich sind. Für beliebiges Polynom $1 * x^{(-)k}$ ist die Ableitung durch $k * x^q$ wo $q = (-)k - 1$ gegeben. Das bedeutet, dass die Eingabe und die Ausgabe aus jeweils 2 Operatoren und 3 Zahlen bestehen.

Kapitel 6

Model

In Rahmen dieses Kapitels wird die Auswahl der Metaparameter für MATH-BERT vorgestellt und begründet. Dazu zählen die Sequenzlänge, die Vokabulargröße, die Batchgröße, die Anzahl an Trainingsepochen und Trainingsbeispielen, die Lernrate, die Schichtenanzahl, die versteckte Größe und der Optimierungsalgorithmus. Ziel ist ein Balance zu finden, so dass keine unnötigen Ressourcen im Anspruch genommen werden. Im Unterkapitel 6.1 sind die drei Modelle zum Untersuchen der ersten Hypothese zu sehen. Anhand diesen werden auch die zweite und die dritte Hypothese getestet.

6.1 Konfigurationen

Die Datenvorbereitung und - repräsentation sind entscheidend für die Sequenzlänge S und die Vokabulargröße V. Für die zwei ausgewählten Datenrepräsentationen - Pre-Order und Layerwise - sind mit Vokabular mit 512 Tokens ungefähr 99% aller vorkommenden Tokens abzudecken. Bezüglich der Sequenzlänge sind mit einer Länge von 256 Tokens ungefähr 95% aller vorkommenden Formeln abzudecken. Aufgrund der hohen Abdeckung werden diese für alle Modelle gleich gehalten. Beim Testen der ersten Hypothese wird mit einer Modellkonfiguration angefangen, die genug Kapazität hat, um die Aufgabe mit einer akzeptablen Genauigkeit zu lösen. Es ist ermittelt worden, dass mit L=4 und H=128 eine Genauigkeit von 80% bei 1 Million Sequenzen erreicht wird. Diese Modellkonfigurationen wird sukzessiv solange vergrößert bis die Batchgröße mindestens 64 Trainingsbeispiele ist. Es wird versucht möglichst viele Datenrepräsentationen (H / D) in Rahmen der Multi-Head-Attention zu generieren und entsprechend wird D möglichst klein gehalten. Die Projektionslänge I wird konstant und größer als die versteckte Größe gehalten. Änderungen werden nur in die Schichtenanzahl und die versteckte Größe gemacht. Es ergeben sich die folgenden Modellkonfigurationen.

MODEL	L	H	I	D	Params	GFLOP
SMALL	4	128	768	4	1.2kk	0.7
BASE	8	256	768	4	6.1kk	3.6
LARGE	12	512	768	4	25.9kk	15

Tabelle 6.1: Math-BERT Modellkonfigurationen

Diese Modelle werden beim Vergleich der beiden Datenrepräsentationen in Rahmen der zweiten Hypothese verwendet. Beim Testen der dritten Hypothese wird nur die Klassifikationsschicht ausgetauscht, so dass sie für die Fine-Tuning-Aufgabe passend ist, aber alle andere Metaparameter bleiben unverändert.

6.2 Training

Vom ersten Blick hat der BERT-ADAM Optimierer nicht einen großen Vorteil gegenüber der Standardvariante vom ADAM. Bezüglich Oszillationen der Ableitung oder Suchen nach lokalen Minimum zeigt er keine eindeutigen Vorteile. Er wird aber benutzt, weil er in NLP-BERT benutzt wird. In allen Modellkonfigurationen wird der Lernrate-Scheduling von BERT genommen. Es wäre aber interessant zu untersuchen, ob das ursprüngliche Scheduling von Transformer nicht zu besseren Ergebnisse führt. Die Autoren von BERT empfehlen die Pre-Training-Phase mit 90.000 Trainingsschritte und Sequenzlänge von 128 Tokens zu beginnen und danach noch weitere 10.000 Trainingsschritte mit Sequenzlänge 512 zu machen, weil so die Trainingsdauer reduziert wird. In Rahmen dieser Masterarbeit wird für das gesamte Training eine konstante Sequenzlänge von 256 Tokens benutzt, denn so werden 95% aller Formeln abgedeckt und eine bessere Datenrepräsentativität gewährleistet, als wenn erst kurze und dann lange Sequenzen genommen werden. Die Google Cloud TPU v2 hat 20 Mal höhere Leistung für 32 Bit Gleitkommazahlen und trotzdem wird 10 Mal länger trainiert. NLP-BERT wird mit 3.3B Tokens, MATH-BERT mit 27B Tokens und GPT-3 mit 500B Tokens trainiert. Im NLP-BERT wird Lernrate von $1e^{-4}$ genommen. Im MATH-BERT wird auf $2e^{-5}$ gesetzt, um Overfitting noch stärker zu dämpfen. Ein Vergleich in der Anzahl an benötigten Trainingsschritten zwischen NLP-BERT und MATH-BERT ist in die folgende Tabelle zu sehen.

Model	Schritte	Dauer	CU	Performanz fp32
NLP-BERT	100,000 [1]	$\leq$ 330 Std.	GCP TPU v2	$\leq$ 180 TFLOPs
MATH-BERT-SMALL	205,078	$\leq$ 192 Std.	1x RTX 5000	$\leq$ 11 TFLOPs
MATH-BERT-BASE	820,312	$\leq$ 550 Std.	1x RTX 5000	$\leq$ 11 TFLOPs
MATH-BERT-LARGE	1,640,625	$\leq$ 1200 Std.	1x RTX 5000	$\leq$ 11 TFLOPs

Abbildung 6.1: Vergleich der Trainingsdauer und Compute-Ressourcen

Kapitel 7

Experimente

In diesem Kapitel werden die vorgestellten Hypothesen anhand der durchgeführten Experimente unter Test gestellt.

7.1 Experiment 1

In Rahmen der ersten Hypothesen wird untersucht, ob die Vergrößerung der Parameteranzahl eine obere Schranke bzgl. der Modellgenauigkeit erreicht. Dafür sind drei Modelle konstruiert. In Tabelle 7.1 ist der Vergleich zwischen diesen zu sehen. Alle sind mit Pre-Order Datenrepräsentation trainiert worden. Mit A1 wird die Genauigkeit beim Vorhersagen der maskierten Tokens und mit A2 beim Einschätzen der Kohärenz bezeichnet.

MODEL	Dauer	Train-A1	Train-A2	Test-A1	Test-A2
SMALL-PRE	8T	0.74	0.79	0.74	0.79
BASE-PRE	24T	0.87	0.83	0.86	0.83
LARGE-PRE	50T	0.89	0.85	0.89	0.85

Tabelle 7.1: Ergebnisse des I. Experimentes

Es ist zu erkennen, dass eine Vergrößerung der Parameteranzahl die Modellgenauigkeit verbessert. Gleichzeitig ist aber auch zu sehen, dass sich die Verbesserung verkleinert, was bedeutet, dass diese langsam eine obere Schranke erreicht, die durch Parametervergrößerung eventuell nicht zu überwinden ist. Während der Unterschied zwischen SMALL-PRE und BASE-PRE in TEST-A1 bzw. TEST-A2 auf 12% bzw. 4% liegt, ist der Unterschied zwischen BASE-PRE und LARGE-PRE nur 3% bzw. 2%. Somit wird angenommen, dass die erste Hypothese bestätigt ist. LARGE-PRE (25kk Parameters) ist 14 Mal kleiner als Google BERT-LARGE (340kk Parameters) und schafft trotzdem nah an eine obere Schranke der Modellgenauigkeit zu kommen.

7.2 Experiment 2

Die zweite Hypothese untersucht die Auswirkung der Datenrepräsentation auf die Modellgenauigkeit. Die Annahme ist, dass die Reihenfolge der Symbole der Formeln entscheidender für die Modellgenauigkeit ist als die Beziehungen zwischen den Symbolen. Um das zu testen, ist das BERT-SMALL Modell mit Pre-Order und Layerwise Daten trainiert worden. Beim Training der beiden Modelle sind gleiche Trainingsparameter und Optimierungsalgorithmen verwendet. Die Ergebnisse sind in die folgende Tabelle zu sehen.

MODEL	Dauer	Train-A1	Train-A2	Test-A1	Test-A2
SMALL-PRE	8T	0.74	0.79	0.74	0.79
SMALL-LAW	8T	0.64	0.78	0.64	0.78

Tabelle 7.2: Ergebnisse des II. Experimentes

Die Resultate bestätigen die zweite Hypothese und auch die Arbeit von Kim et al. (2020), dass Pre-Order bei Baumstrukturen vorteilhafter sein kann.

Die Layerwise-Datenrepräsentation zeigt in TEST-A1 10% und in TEST-A2 1% kleinere Modellgenauigkeit als die Pre-Order-Datenrepräsentation. Eine Erklärung für diesen Unterschied ist das doppelte Vorkommen von Knoten in Layerwise. Bei Layerwise kommt die Kodierung von Knoten, die gleichzeitig Eltern- und Kinderknoten sind, doppelt vor, weil diese in zwei Beziehungen involviert sind. Das ist einerseits vom Vorteil, weil so das Modell besser auf die Beziehungen der Knoten fokussiert wird. Allerdings scheint nicht eine Auswirkung auf die Modellgenauigkeit zu haben.

Eine weitere Erklärung liegt in das Layout von MathML. MathML speichert die Knoten von Funktionen und Operatoren auf die gleiche Schicht wie die Argumente. Dementsprechend ist es sowohl für Layerwise als auch für Pre-Order schwieriger Beziehungen zu erkennen. Da aber Layerwise stärker auf Beziehungen fokussiert ist und wegen des Layouts davon nicht so stark profitieren kann, kann das auch ein Grund für den Unterschied in die Modellgenauigkeiten sein.

Aufgrund des großen Unterschieds in die Modellgenauigkeiten ist das Experiment nicht mit BERT-BASE und BERT-LARGE durchgeführt, weil es angenommen wird, dass da Pre-Order auch eine bessere Modellgenauigkeit erreichen würde.

7.3 Experiment 3

In Rahmen der dritten Hypothese wird untersucht, ob das Pre-Training zur verbesserten Genauigkeit beim Fine-Tuning führt. Die Frage ist, ob ein grundlegendes Verständnis erschaffen werden kann, so dass weitere mathematische Aufgaben davon profitieren können, auch wenn für diese wenig Daten zur Verfügung stehen. In Rahmen dieses Experimentes ist jedes der drei vorgestellten Modelle jeweils für die diskriminative und generative Aufgabe zweimal trainiert worden. Einmal pre-trained (PRE) und danach nicht pre-trained (NO-PRE). Beim Fine-Tuning von pre-trained Modellen ist bei der diskriminativen Aufgabe nur die Klassifikationsschicht und bei der generativen alles optimiert. Dabei wurde mit verschiedenen Lernraten für jeweils 100 Epochen trainiert und daraus das beste Resultat genommen. In die folgende Tabelle sind die Ergebnisse der diskriminativen Aufgabe zu sehen:

MODEL	Train-L1	Test-L1	Train-A1	Test-A1
SMALL-PRE	2.40	3.30	0.94	0.82
SMALL-NO-PRE	0.93	25.1	1.0	0.5
BASE-PRE	0.40	0.01	1.0	1.0
BASE-NO-PRE	0.83	40.0	1.0	0.64
LARGE-PRE	0.80	2.60	1.0	0.88
LARGE-NO-PRE	0.92	38.1	1.0	0.57

Tabelle 7.3: Ergebnisse des III.a Experimentes

Jedes der Modelle hat genug Kapazität um die Aufgabe zu erlernen, da eine 1.0 Genauigkeit beim Train-A1 leicht zu erreichen ist. Die Unterschiede kommen beim Testen. Bei den pre-trained Modellen korreliert die Train-A1 stärker mit der Test-A1, was bedeutet, dass diese Modelle schwieriger zum Overfitting kommen und bessere Generalisierung der Aufgabe erreichen.

Beim Versuch Polynome generativ abzuleiten werden die zwei Kriterien Gültigkeit und Korrektheit gemessen. Erst wird geprüft, ob der generierte Ausdruck ein gültiger mathematischer Ausdruck ist, und dann, ob er die mathematisch korrekte Ableitung des Eingabepolynoms ist. Die Gültigkeit wird geprüft, indem der generierte Ausdruck zurück zu MathML dekodiert wird. Die Korrektheit wird durch Messung des Abstands zu der richtigen Ableitung ermittelt. Das bedeutet, dass je näher die geschätzte Kodierung ist, desto großer wird die Genauigkeit sein. Zum Beispiel bei richtiger Ableitung x^{10} wird die Schätzung x^9 eine bessere Genauigkeit aufweisen als x^8. Dieses Kriterium hat den Vorteil, dass er nicht nur zwischen richtig und falsch unterscheidet. Es ergeben sich die folgenden Resultate:

MODEL	Train-L1	Test-L1	Train-A1	Train-Gültig	Test-A1	Test-Gültig
SMALL-PRE	3.11	10.31	0.9892	1.0	0.815	0.970
SMALL-NO-PRE	3.86	12.25	0.996	1.0	0.829	0.101
BASE-PRE	3.37	10.72	0.993	1.0	0.860	0.960
BASE-NO-PRE	4.90	12.87	0.996	1.0	0.830	0.400
LARGE-PRE	3.72	10.91	0.995	1.0	0.850	0.945
LARGE-NO-PRE	11.3	13.39	0.390	0.4	0.358	0.304

Tabelle 7.4: Ergebnisse des III.b Experimentes

Das Pre-Training hat auf die TEST-A1 (die mathematische Genauigkeit) einen erkennbaren Einfluss. Bei den BASE-Modellen führt das Pre-Training zu 4% höhere Genauigkeit und bei LARGE zu 50%. Bei SMALL hat das NO-PRE Modell eine bessere Genauigkeit, aber um 80% schlechtere Gültigkeit. Die höchste Genauigkeit von allen Modellen ist bei BASE-PRE. Der größte Unterschied zwischen PRE und NO-PRE ist in die Gültigkeit. Da schaffen die pre-trained Modelle eine zwischen 50% und 80% höhere Gültigkeit und kommen nah an 100%. Das liegt daran, dass während des Pre-Trainings die Modelle ein grundlegendes Verständnis für die Struktur von mathematischen Ausdrücken erlernen, was ein sehr effektiver Vorteil bei der generativen Aufgabe ist. Die Top 10 TEST-A1-Ergebnisse von BASE-PRE sind in die folgende Tabelle zu sehen:

No.	Inference	Target	Accuracy
20	$38x^{37}$	$38x^{37}$	1.0000
14	$112x^{112}$	$112x^{111}$	0.9980
23	$224x^{224}$	$224x^{223}$	0.9980
31	$154x^{154}$	$154x^{153}$	0.9980
41	$72x^{72}$	$72x^{71}$	0.9980
48	$37x^{37}$	$37x^{36}$	0.9980
49	$210x^{210}$	$210x^{209}$	0.9980
70	$54x^{54}$	$54x^{53}$	0.9980
103	$179x^{179}$	$179x^{178}$	0.9980
108	$127x^{127}$	$127x^{126}$	0.9980

Abbildung 7.1: Geschätzte Ableitung, richtige Ableitung und Genauigkeit

Kapitel 8

Ausblick

In diesem Kapitel werden eine Evaluation der Ergebnisse und einige potentielle Verbesserungen vorgestellt und diskutiert.

8.1 Evaluation

Transformer-Modelle und insbesondere BERT eignen sich gut zum Erlernen vom mathematischen Verständnis, denn die besten Ergebnisse in Rahmen des Pre-Trainings zeigen eine Genauigkeit von 89% bei der Einschätzung von maskierten Tokens und 85% bei der Vorhersage der Kohärenz von Formeln. Allerdings kommt das größte Modell - BERT-LARGE - an eine obere Schranke der Verbesserung. Das bedeutet, dass eine Parametervergrößerung nicht immer die Lösung zur Erhöhung der Modellgenauigkeit ist. Für BERT ist schwieriger die Kohärenz zu schätzen, weil bei dieser eine größere Variation in die Eingabemöglichkeiten besteht als bei der Maskierung von Tokens, weil die Eingabe zwei beliebige Formeln fusioniert, aber die maskierten Tokens nur aus dem Vokabular sind. Es stellt sich fest, dass sich mathematische Ausdrücke sehr effizient repräsentieren lassen, da mit einem kleinen Vokabular und einer kurzen Sequenzlänge fast alle Tokens und Formeln aus dem arXiv-Datensatz darstellbar sind. Außerdem ist die Reihenfolge von Symbolen entscheidender für die Modellgenauigkeit als die Beziehungen zwischen den Symbolen. In Rahmen des Fine-Tunings ist bestätigt worden, dass das Pre-Training vom Vorteil ist, da dadurch ein grundlegendes Verständnis erlernt wird, welches anhand von wenig Daten und Trainingsaufwand in Rahmen des Fine-Tunings nicht zu erlernen ist. Hier erzielt das pre-trained BERT-BASE die besten Ergebnisse mit einer Test-Genauigkeit von 100% bei der diskriminativen und 86% bei der generativen Aufgabe. Nicht pre-trained Modelle mit genug Kapazität zum Erlernen der Aufgabe erreichen mit wenig Daten keine hinreichende Generalisierung beim Testen und kommen schnell zum Overfitting. Das Pre-Training führt außerdem zu einer besseren Gültigkeit bei der Generierung.

8.2 Verbesserungen

Die Datenrepräsentation ist entscheidend für die Modellgenauigkeit und entsprechend ist die Suche nach alternativen Datenrepräsentationen nicht abgeschlossen. Es soll weiter in diese Richtung geforscht und experimentiert werden, weil es nicht auszuschließen ist, dass es eine bessere Datenrepräsentation als Pre-Order existiert. Die Verwendung von noch besseren Datenrepräsentationen kann dazu führen, dass die Modelle dieselbe Aufgaben mit kleinerem Trainingsaufwand und kleinerer Parameteranzahl erlernen können.

Die Vergrößerung der Modellparameter durch mehr Schichten und größere versteckte Größe haben direkten Einfluss auf die Genauigkeit, aber diese scheint eine obere Schranke zu erreichen. Das bedeutet, dass ab einem bestimmten Punkt nur eine bessere Modellarchitektur oder Datenrepräsentation eine Lösung zur Erhöhung der Modellgenauigkeit sein kann. Alternative Modellarchitekturen sollen auch weiter erforscht werden.

BERT, GPT-3 und Switch-C folgen der Annahme, dass die Modellgenauigkeit durch Parametervergrößerung nicht nach oben beschränkt ist. GPT-3 macht den Schritt auf 175 Milliarden und Google in Fedus et al. (2021) auf mehr als 1 Trillion Parameter. Intentionen auf weitere Vergrößerungen sind nicht ausgeschlossen. Die Überzeugung dieser Masterarbeit ist, dass sich die Forschung stärker auf die Modifikation der modernen Modellarchitekturen und das Suchen nach neuen Schichttypen fokussieren soll. Zum Beispiel die Suche nach einer Alternative zu Multi-Head-Attention, weil diese die Beziehung jeder zwei Tokens der Sequenz betrachtet, aber es kann sein, dass längere Beziehungsketten bedeutender sind - also zum Beispiel 3 oder mehr Tokens.

Die durchgeführten Experimente zum Fine-Tuning der Modelle zeigen, dass das Pre-Training sowohl bei diskriminativen als auch bei generativen Aufgaben vom Vorteil sein kann. Es soll untersucht werden, wie flexible das Pre-Training ist, wenn beim Fine-Tuning große Abweichungen von den erlernten Datenrepräsentationen und Aufgaben bestehen.

Abbildungsverzeichnis

Literaturverzeichnis

Dario Amodei, Rishita Anubhai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Jingdong Chen, Mike Chrzanowski, Adam Coates, Greg Diamos, Erich Elsen, Jesse H. Engel, Linxi Fan, Christopher Fougner, Tony Han, Awni Y. Hannun, Billy Jun, Patrick LeGresley, Libby Lin, Sharan Narang, Andrew Y. Ng, Sherjil Ozair, Ryan Prenger, Jonathan Raiman, Sanjeev Satheesh, David Seetapun, Shubho Sengupta, Yi Wang, Zhiqian Wang, Chong Wang, Bo Xiao, Dani Yogatama, Jun Zhan, and Zhenyao Zhu. Deep speech 2: End-to-end speech recognition in english and mandarin. *CoRR*, abs/1512.02595, 2015. URL http://arxiv.org/abs/1512.02595.

Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations, 2020.

Emily Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. On the dangers of stochastic parrots: Can language models be too big?, 2021.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity, 2021.

Glubenchmark. Glubenchmark, 2020. URL https://gluebenchmark.com/leaderboard. last accessed on March 24, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017. URL http://arxiv.org/abs/1703.06870.

Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus), 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

Seohyun Kim, Jinman Zhao, Yuchi Tian, and Satish Chandra. Code prediction by feeding trees to transformers, 2020.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

Guillaume Lample and François Charton. Deep learning for symbolic mathematics. *CoRR*, abs/1912.01412, 2019. URL http://arxiv.org/abs/1912.01412.

MinZhong Luo and Li Liu. Automatic derivation of formulas using reforcement learning. *CoRR*, abs/1808.04946, 2018. URL http://arxiv.org/abs/1808.04946.

Tomas Mikolov, Ilya Sutskever, Kai Chen, G.s Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems*, 26, 10 2013.

Lukas Pfahler and Katharina Morik. Semantic search in millions of equations. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2020*, 2020.

Lukas Pfahler, Jonathan Schill, and Katharina Morik. The search for equations – learning to identify similarities between mathematical expressions. 2019.

Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018. URL http://arxiv.org/abs/1804.02767.

Anna Rogers, Olga Kovaleva, and Anna Rumshisky. A primer in bertology: What we know about how bert works, 2020.

SQuAD2. Squad2, 2020. URL https://rajpurkar.github.io/SQuAD-explorer/. last accessed on March 27, 2020.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. *CoRR*, abs/1808.01974, 2018. URL http://arxiv.org/abs/1808.01974.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762.