HAW HAMBURG

BACHELORTHESIS
André Soblechero Salvado

Text-Klassifikation durch BERT-basiertes Text-Splitting gesteuert durch einen Suchalgorithmus

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

André Soblechero Salvado

Text-Klassifikation durch BERT-basiertes Text-Splitting gesteuert durch einen Suchalgorithmus

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Michael Neitzke
Zweitgutachter: Prof. Dr. Stephan Pareigis

Eingereicht am: 23. Januar 2020

André Soblechero Salvado

Thema der Arbeit

Text-Klassifikation durch BERT-basiertes Text-Splitting gesteuert durch einen Suchalgorithmus

Stichworte

BERT, Text-Klassifikation, Dokumenten-Klassifikation, Textsplitter, Satzsegmentation, Gated Recurrent Units, Algorithmus, Transformer

Kurzzusammenfassung

Diese Arbeit stellt eine Möglichkeit vor trainierte Text-Klassifikatoren zu verbessern. Dies wird ermöglicht indem nach dem Textsegment mit der höchsten Wahrscheinlichkeit für eine Klasse gesucht wird. Der Algorithmus konzentriert sich auf umgangssprachliche Texte und nutzt BERT, ein vortrainiertes Sprachmodell, welches im Jahr 2018 die wichtigsten Computerlinguistik-Bestenlisten dominierte.[4]

André Soblechero Salvado

Title of Thesis

Text-Classification by BERT-based text-splitting controlled by a searchalgorithm

Keywords

BERT, text-classification, documentclassification, textsplitter, sentence segmentation, gated recurrent units, algorithm, transformer

Abstract

This paper introduces a new way to improve your trained text-classifier by searching for the text-segment with the highest likelihood for a class instead of using the whole text at once. This algorithmen focuses on colloquial speech and uses BERT, a pretrained language model, which domniated the most important nlp benchmarks for some time in 2018.[4]

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Computerlinguistik ist ein Forschungsgebiet der Informatik, welches die Kommunikation zwischen Mensch und Maschine untersucht. Die Forschung auf diesem Gebiet ist bedeutend, da Sprache der Schlüssel zur Kommunikation mit Menschen ist. Mithilfe der Sprache können Menschen am gesellschaftlichen Leben teilnehmen und Tätigkeiten ausüben. Zur Automatisierung von Tätigkeiten bedarf es dementsprechend der Forschung auf dem Gebiet der Computerlinguistik.

Der Bereich der Computerlinguistik hat in den letzten Jahren dank „deep learning“ und immer leistungsfähigerer Hardware große Fortschritte machen können. Speziell die Bereiche, die sich der Klassifizierung und dem generieren von Texten widmen, konnte enorm profitieren.

Diese Arbeit stellt eine Möglichkeit vor trainierte Text-Klassifikatoren zu verbessern. Dies wird ermöglicht indem nach dem Textsegment mit der höchsten Wahrscheinlichkeit für eine Klasse gesucht wird. Der Algorithmus konzentriert sich auf umgangssprachliche Texte und nutzt BERT, ein vortrainiertes Sprachmodell, welches im Jahr 2018 die wichtigsten Computerlinguistik-Bestenlisten dominierte.[4]

1.1 Persönliche Motivation

Die persönliche Motivation an der Arbeit in der Computerlinguistik entspringt primär meiner Meinung, dass eine starke künstliche Intelligenz zwingend Sprache benötigt, um mit Menschen kommunizieren zu können. Somit ist Computerlinguistik die Prämisse, um künstliche Intelligenz als stark einzustufen. Fortschritte wie GPT2 zeigen, dass Modelle des maschinellen Lernens bereits dazu imstande sind komplexe Texte zu generieren und sich außerordentlich viele Informationen über Sprache beizubringen. [13] Das Thema der Textklassifizierung hingegen ist ein Ansatz, welcher heute bereits in der Wirtschaft produktiv eingesetzt werden kann wie beispielsweise in Chatbots oder Hassredenerkennung im Internet. Dies zeigt auf, dass dieses Feld weiterhin am Wachsen ist und in Zukunft interessant bleiben wird. Gleichzeitig wird auch deutlich, welche Verantwortung solch ein Text-Klassifikator haben kann. Text-Klassifikatoren entscheiden somit auch ob ein Text-Beitrag zensiert wird oder nicht, was ethisch und rechtlich nicht immer eindeutig ist.

Für manche Problemstellungen werden Texte vor der Klassifizierung mit einem Textsplitter, welcher auf regulären Ausdrücken basiert, separiert, um einen Anfragetext in Teilprobleme zu zerlegen. Die Klasse mit der höchsten Wahrscheinlichkeit wird dann auf den gesamten Text angewendet.

Mein ursprüngliches Ziel war es eine Alternative zur Satzsegmentierung durch reguläre Ausdrücke zu erarbeiten, da reguläre Ausdrücke eine korrekte Semantik und Syntax voraussetzen, um eine optimale Separierung zu gewährleisten, welche nicht immer gegeben sind.

1.2 Text-Klassifikation

Die Text-Klassifikation behandelt die Problematik eine Folge von Tokens bzw. Wörtern, Wortteilen oder Ziffern, einer vordefinierten Klasse zuzuordnen.

$$c : S \rightarrow C \tag{1.1}$$

D = {Alle Tokens: Wörter, Teilwörter und Ziffern, die im Trainingsdatensatz vorkommen}
S = {x | x ist Element der Menge aller möglichen Texte, welche aus der Menge D konstruiert werden können}
C = {Menge aller verfügbaren Klassen}

Somit ist dies die Definition der Funktion c, welche einen Text entgegennimmt und eine Klasse zurückgibt.

Gegeben seien beispielsweise die Klassen C={Aussage,Frage} sowie die Folge von Tokens „Mein Name ist André!“. Die korrekte Klasse wäre intuitiv betrachtet „Aussage“, da der Satz offensichtlich eine Aussage ist. Im maschinellen lernen muss diese Korrelation c(„MeinNameistAndré!“)=„Aussage“ dem zu trainierenden Modell anhand von Trainingsdaten erst beigebracht werden.

Manche Modelle erlauben es eine von dem Modell selbst beigebrachte Wahrscheinlichkeit für die Klasse auszugeben, weswegen wir nun die Ableitung 1.1 anpassen.

$$c : S \rightarrow C \tag{1.2}$$

D = {Alle Tokens: Wörter, Teilwörter und Ziffern, die im Trainingsdatensatz vorkommen}
S = {x | x ist Element der Menge aller möglichen Texte, welche aus der Menge D konstruiert werden können}
C = {(y, z) | y ∈ {Menge aller verfügbaren Klassen} und z ∈ [0,1]}

1.3 Transferlernen

Transferlernen im maschinellen Lernen beschreibt das Trainieren eines Modells auf eine Aufgabe im ersten Schritt und die Feinabstimmung des Modells auf ggf. eine andere Aufgabe im zweiten Schritt, um von dem Wissen, dass sich das Modell im ersten Schritt angeeignet hat, im zweiten Schritt zu profitieren.

Transferlernen kommt aus der computerunterstützten Bildverarbeitung, genauer dem „deep learning“, wo zunächst ein Modell auf ImageNet, einem riesigen Datensatz für die Klassifizierung von Bildern mit Klassen wie „Pferd“ oder „Auto“, trainiert wurde und daraufhin die Modelle feinabgestimmt wurden auf andere Aufgaben. Diese Methodik führte zu deutlich besseren Ergebnissen. [23]

1.4 Neuronale Sprachmodelle

Neuronale Sprachmodelle sind aktuell die Grundlage für die meisten Fortschritte in der Computerlinguistik. Viele neuronale Sprachmodelle werden dahingehend trainiert als Output die Wörter zu nennen, welche in einem unvollständigen Satz fehlen. Dies hat den Effekt, dass das Modell Abhängigkeiten von Wörtern lernt und sich somit die Logik einer Sprache implizit beibringt. Diese vortrainierten Sprachmodelle werden verwendet, um beispielsweise Word-Embeddings zu generieren. Word-Embeddings sind mathematische Darstellungen für Tokens, sowie deren Bedeutung im Text, in Form von Vektoren.

Ebenfalls sind diese Sprachmodelle feinabstimmbar, was heißt, dass diese vortrainierten Modelle und deren Parameter durch Trainings dahingehend verändert werden speziellere Aufgaben zu erfüllen, statt lediglich fehlende Wörter vorherzusagen. Das Ziel ist identisch mit dem des Transferlernens. Im ersten Schritt wird dem Modell zunächst die Sprache beigebracht, damit im zweiten Schritt dem Modell eine spezielle Aufgabe antrainiert werden kann, wie beispielsweise das Klassifizieren von Texten.

Die Bedeutung der Sprachmodelle lässt sich aus den allgegenwärtigen Problemen ableiten wenige Daten zum Trainieren eines Modells auf spezielle Aufgaben zur Verfügung zu haben und die Kosten bzw. Dauer von Trainings gering zu halten. Sprachmodelle lassen sich ohne vom Menschen klassifizierte Daten, welche teuer sind, unüberwacht trainieren. XLNet-Large, eine Weiterentwicklung des BERT-Sprachmodells, welches in dieser Arbeit

nicht genauer beleuchtet wird, wurde mit 512 Tensor Processing Units v3 Chips über 2,5 Tage trainiert. Diese Zahlen verdeutlichen, dass Kosten und Dauer enorme Faktoren sind, wenn es darum geht optimale Modelle zu trainieren. [22]

1.5 Problemstellung

Das Ziel Texte in Teile zu zerlegen, um eine Vereinfachung der Klassifizierung zu ermöglichen wurde bisher meist mit regulären Ausdrücken sowie einem gewöhnlichen Klassifikator erreicht. Reguläre Ausdrücke benötigen dabei semantisch und syntaktisch korrekte Texte und müssen zahlreiche Ausnahmen, beispielsweise Abkürzungen wie „etc.“, kennen. Problematisch dabei ist, dass Texte im Internet häufig umgangssprachlicher Natur und somit syntaktisch und semantisch nicht zwingend korrekt sind. Dies erschwert den Einsatz von regulären Ausdrücken erheblich.

Ein weiteres Problem ist, dass Menschen manchmal ihre Absicht/Klasse in Texten in beispielsweise 3 Wörter äußern und um diese 3 Wörter nur Fülltext vorhanden ist, welcher das Modell verunsichern kann.
Folgender Anwendungsfall verdeutlicht die Problematik.
Ein Unternehmen bietet eine Chatbot-Lösung an. Für die Lösung muss es einen Service geben, welcher Texte antrainierten Absichten/Klassen zuordnet.
Es gibt die Absichten C = {Discount, Pricing, Window Information}
Der Satz “At first we where thinking about regular windows but then our architect recommended us to have a look at yours, and now we are wondering what would be the total cost of including them into out project?” soll einer Absicht zugeordnet werden.

Auffallend ist, dass im Grunde genommen nur der Textteil „what would be the total cost“ für den Klassifikator wichtig ist, um den Satz der Klasse Pricing zuzuordnen, jedoch sind um diesen Textteil noch viele Füllwörter, welche zu einer Verunsicherung, in Form von einer zu geringen Wahrscheinlichkeit oder Falschklassifizierung des Textes führen könnten. Des Weiteren sind reguläre Ausdrücke zum zerteilen dieses Textes in Teilprobleme nicht anwendbar, da keine Zeichensetzung beachtet wurde.

Es stellt sich die Frage, wie Texte so zerteilt werden könnten, so dass für die Klassifizierung unwichtige Textteile ignoriert werden.

2 BERT Textsplitter Klassifikationsalgorithmus

In diesem Abschnitt wird ein Algorithmus zur Verbesserung von Text-Klassifikatoren vorgestellt, welcher die Genauigkeit des Text-Klassifikators gegenüber dem alleinstehenden Text-Klassifikator um bis zu 10% in einer problemspezifischen Accuracy und einem problemspezifischen F1-Score verbessern kann und auf einem Text-Splitter sowie der Breitensuche basiert.
Zunächst muss dafür jedoch der Textsplitter definiert werden.

$$s : S \to P \tag{2.1}$$

D = {Alle Tokens: Wörter, Teilwörter und Ziffern, die im Trainingsdatensatz vorkommen}
S = {x | x ist Element der Menge aller möglichen Texte, welche aus der Menge D konstruiert werden können}
P = {(y, z) | y ist ungleich z, y und z zerteilen den Text x optimal, yz = x}

Die Definition eines optimal geteilten Textes ist in Abschnitt 3.1 gegeben. Somit ist dies die Funktion, die einen Text entgegennimmt und diesen in zwei Texte zerteilt. s(„Mein Name ist André und ich schreibe meine Bachelorarbeit“) müsste infolgedessen den Text in („Mein Name ist André“, „und ich schreibe meine Bachelorarbeit“) zerteilen.

Ebenfalls muss definiert werden, ab wann die Wahrscheinlichkeit einer Klassifikation hoch genug ist um als Voraussage zu gelten bzw. ab wann die Voraussage aufgrund von Unsicherheit als False Negative gewertet wird.

c(„Mein Name ist André“) = (Aussage, 0.1) wäre beispielsweise zu unsicher als dass behauptet werden könne, dass dies ein True Positive ist.

c(„Mein Name ist André“) = (Aussage, 0.9) wäre eine auswertbare Voraussage, da die Wahrscheinlichkeit für die Klasse einen Schwellwert von 0.5 überschreitet.

Im Folgenden wird der Schwellwert für die Voraussage von Beispielen auf 0.5 festgelegt. In den Experimenten wird nach einem Schwellwert gesucht, welcher die Accuracy/F1 Score gegen ein Maximum konvergieren lässt.

2.1 Algorithmus

Der Algorithmus besteht aus einer Breitensuche, welche den Text zerteilt und jeweils die Textteile klassifiziert. Anhand der Klassifizierung wird nach einer Klasse mit einer Wahrscheinlichkeit gesucht, die den Schwellwert von 0.5 überschreitet, um diese für den gesamten Text geltend zu machen.

1. Füge den Text der Warteschlange hinzu
2. Überprüfe ob ein Element in der Warteschlange ist
   a) Falls etwas in der Warteschlange ist, dann nehme das Element x heraus und gehe zu Punkt 3.
   b) Ansonsten gehe zu Punkt 5.
3. Überprüfe ob c(x) eine Klasse mit einer Wahrscheinlichkeit größer gleich dem Schwellwert voraussagen kann
   a) Wenn ja gebe die Klasse sowie die Wahrscheinlichkeit zurück
   b) Ansonsten gehe zu Punkt 4.
4. Überprüfe ob der aktuelle Text x aus einem Wort besteht
   a) Wenn ja gehe zu Punkt 2.
   b) Ansonsten zerteile den Text in 2 Texte mit s(x), füge die beiden Elemente des resultierenden Tupels jeweils der Warteschlange hinzu und gehe zu Punkt 2.
5. Suche unter allen vorausgesagten Klassen und den korrespondierenden Wahrscheinlichkeiten die Klasse mit der höchsten Wahrscheinlichkeit und gebe diese zurück

2.2 Veranschaulichung

Folgendes Beispiel, welches den Anwendungsfall aus der Einleitung verwendet, verdeutlicht den Algorithmus.

Absichten/Klassen:

C = {(y, z) | y ∈ Discount, Pricing, Window Information und z ∈ [0,1]}
Zu klassifizierender Satz:
"At first we where thinking about regular windows but then our architect recommended us to have a look at yours, and now we are wondering what would be the total cost of including them into out project?"

Der binäre Baum in der Abbildung 2.1 visualisiert die Breitensuche nach einer Klassifikation, welche eine Wahrscheinlichkeit größer dem Schwellwert hat. Ebenfalls visualisiert ist die Warteschlange, welche die Reihenfolge der Klassifikationen der Texte bzw. Textteile wiederspiegelt. Anhand der Farbe können die einzelnen Klassifikationen mit dem dazugehörigen Text in Verbindung gebracht werden. Die Reihenfolge entspricht einer Breitensuche und kann an den Kanten abgelesen werden.

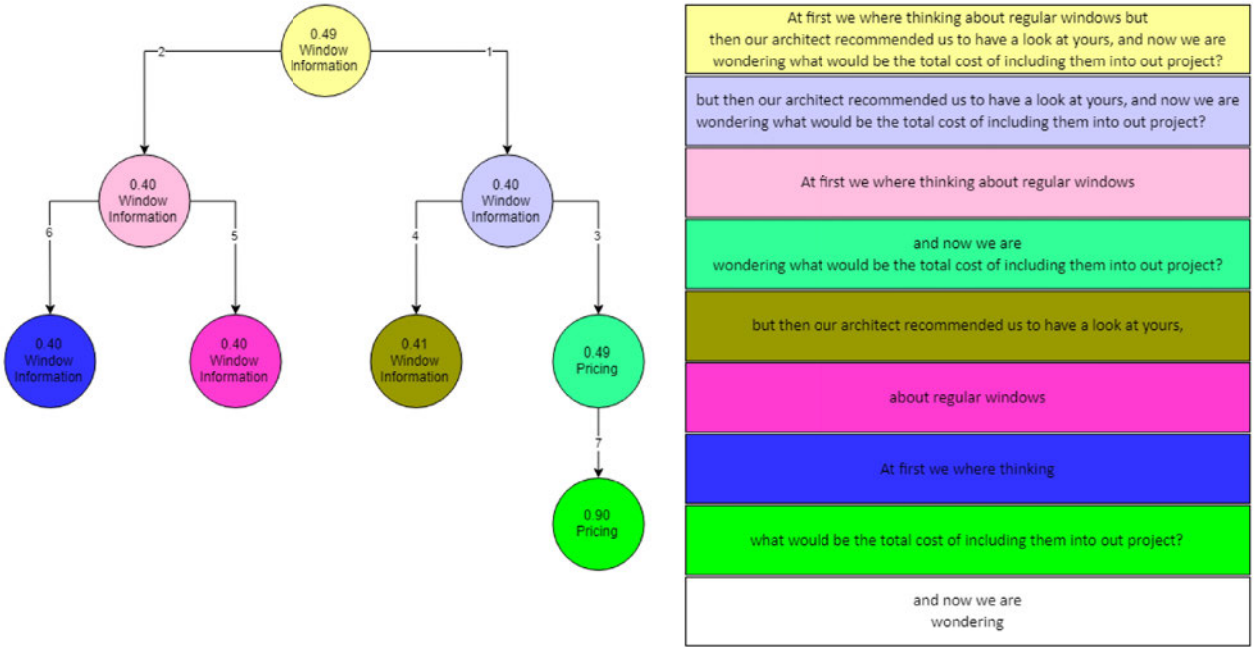

Abbildung 2.1: Veranschaulichung der Breitensuche

Beginnend mit der Klassifikation des ursprünglichen Textes in der Wurzel, welcher mit einer Wahrscheinlichkeit von 0.49 der Klasse „Window Information“ zugeordnet wird. Diese Wahrscheinlichkeit ist geringer als der Schwellwert und somit wird der ursprüngliche Text geteilt und die resultierenden Textteile werden der Warteschlange hinzugefügt.

Die Kinder der Wurzel sind die Klassifikationen der jeweiligen Textteile, welche zuvor durch das Teilen des ursprünglichen Textes erstellt werden. Dieser Prozess wird solange wiederholt bis ein Text mit einer Wahrscheinlichkeit von 0.90 der Klasse „Pricing“ zugeordnet wird. Diese Klasse wird nun für den ursprünglichen Text verwendet und der Algorithmus ist beendet. Aufgrund dessen wird der letzte Eintrag in der Warteschlange nicht mehr beachtet.

3 BERT Textsplitter

Der Kern des Algorithmus ist der BERT Textsplitter. BERT ist ein Sprachmodell, welches mitunter die bisher größten Durchbrüche in den letzten Jahren in der Computerlinguistik hervorbrachte. Der Textsplitter nimmt einen Text x entgegen und gibt 2 Texte y und z zurück, wo x = yz und die Trennung des Textes optimal ist.

$$s: \ S \rightarrow P \tag{3.1}$$

D = {Alle Tokens: Wörter, Teilwörter und Ziffern, die im Trainingsdatensatz vorkommen}
S = {x | x ist Element der Menge aller möglichen Texte, welche aus der Menge D konstruiert werden können}
P = {(y, z) | y ist ungleich z, y und z zerteilen den Text x optimal, yz = x}

3.1 Algorithmus

1. Speicherung jeder möglichen Teilung des Textes x als Textpaar.
2. Ermittlung der Wahrscheinlichkeit inwieweit ein Textpaar aufeinander folgt durch das vortrainierte Sprachmodell BERT.
3. Das Textpaar mit der geringsten Wahrscheinlichkeit wird vom Algorithmus zurückgegeben.

Somit ist eine optimale Trennung eines Textes definiert durch eine Wahrscheinlichkeit, die bestimmt inwieweit zwei Texte aufeinander folgen.

Folgender Text soll gesplittet werden:

1. ”Ich bin Peter und ich mag Spiele”

Es würden folgende Texte generiert werden:

1. "[CLS] Ich [SEP] bin Peter und ich mag Spiele [SEP]"
2. "[CLS] Ich bin [SEP] Peter und ich mag Spiele [SEP]"
3. "[CLS] Ich bin Peter [SEP] und ich mag Spiele [SEP]"
4. "[CLS] Ich bin Peter und [SEP] ich mag Spiele [SEP]"
5. "[CLS] Ich bin Peter und ich [SEP] mag Spiele [SEP]"
6. "[CLS] Ich bin Peter und ich mag [SEP] Spiele [SEP]"

Der [SEP]-Token symbolisiert das Ende einer Folge von Tokens.

Der [CLS]-Token dient als Vektor, auf welchen Klassifikationen des gesamten Textes ausgeführt werden. Genauere Erläuterungen folgen im Abschnitt 4.

Wenn nun alle Sätze an das BERT Modell weitergegeben werden bestimmt dieses Modell für jeden Satz eine reelle Zahl, welche aussagt wie wahrscheinlich es ist, dass die beiden Folgen von Tokens aufeinander folgen.

1. ”[CLS] Ich [SEP] bin Peter und ich mag Spiele [SEP]” - 0.5
2. ”[CLS] Ich bin [SEP] Peter und ich mag Spiele [SEP]” - 0.4
3. ”[CLS] Ich bin Peter [SEP] und ich mag Spiele [SEP]” - 0.1
4. ”[CLS] Ich bin Peter und [SEP] ich mag Spiele [SEP]” - 0.2
5. ”[CLS] Ich bin Peter und ich [SEP] mag Spiele [SEP]” - 0.5
6. ”[CLS] Ich bin Peter und ich mag [SEP] Spiele [SEP]” - 0.6

Der Text mit der geringsten Wahrscheinlichkeit ist in diesem Beispiel der Satz ”[CLS] Ich bin Peter [SEP] und ich mag Spiele [SEP]”.

4 BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Das BERT Sprachmodell ist der zuletzt größte Durchbruch in der Computerlinguistik. Auf diesem Sprachmodell basieren die aktuellsten Computerlinguistik Modelle, wie RoBERTa oder XLNet, die in weit verbreiteten Benchmarks wie SuperGlue zu den besten Modellen gehören. [22] [19] [9] [4]

SuperGlue ist eine Sammlung von 9 verschiedenen Computerlinguistik-Benchmarks, welche unter anderen das Sprachverständnis von Modellen auf öffentlichen Datensätzen testet.

Nun stellt sich die Frage wie ein Modell trainiert wird, welches aussagen kann, inwieweit zwei Texte aufeinander folgen, da es vermutlich keine Datensätze gibt, welche umgangssprachlich sind und zudem eine optimale Trennung eines Textes angeben.

Umgangssprachliche Datensätze werden benötigt, da bei grammatikalisch korrekten Texten ein auf regulären Ausdrücken basierender Textsplitter vollkommen ausreichend ist.

Da durch das Nachvollziehen von BERT die Idee zum BERT Textsplitters erst entstand wird in diesem Kapitel BERT und die besondere Art des Trainings dieses Sprachmodelles im Detail erläutert.

BERT steht für "Bidirectional Encoder Representations from Transformers". Das "Bidirectional" entspringt der Tatsache, dass BERT nicht wie herkömmliche rekurrente Netze von links nach rechts oder von rechts nach links sich Wissen über den Text aneignet, bzw. Texte nicht sequenziell auswertet, sondern den gesamten Text als Ganzes verarbeitet.

Der Vorteil, der sich daraus ergibt, ist eine erhöhte Parallelisierung und somit eine effektivere Ausnutzung von GPUs oder TPUs, wie es bei der Bildverarbeitung bereits der Fall ist. Der genaue Grund für die Verwendung von "Bidirectional" wird nach genauerem Betrachten der Architektur und des Trainings ersichtlich.
"Transformer" steht für die von BERT verwendete Architektur, welche im Paper "Attention is all you need" vorgestellt wurde und im Folgenden ebenfalls genauer erläutert wird.[18]

4.1 Modelleingabe

Die Eingabe für das Sprachmodell und die Begründung für die Wahl dieser Eingabe-Formen wird in diesem Abschnitt expliziter erläutert.
Ein Problem der Computerlinguistik speziell in Kombination mit maschinellem Lernen ist das handhaben von Tokens, welche nicht im Trainingsdatensatz vorkommen und somit den jeweiligen Modellen unbekannt sind. Bislang war es meist so, dass unbekannte Tokens genau einem generischen Token Embedding zugeordnet wurden. Diese Lösung ist nicht optimal, da das Modell keine Informationen über den Token selbst erhält und nur versuchen kann anhand der umgebenen Wörter zu erraten was der jeweilige unbekannte Token sein kann bzw. welche Aussagekraft er hat. Ebenfalls ist es so, dass diese Modelle nie zuvor das Token-Embedding von unbekannten Tokens gesehen haben, da sie nur die im Trainingsdatensatz vorkommenden Wörter kennen.
Ein weiteres Problem entsteht durch die parallele Verarbeitung von Tokens, da keine Information über die Position des Tokens im Text vorhanden ist, wie das bei rekurrenten Netzwerken implizit der Fall ist. Dieses Problem handhabt BERT mit sogenannten Position Embeddings.
Durch die Segment Embeddings ist es möglich zwei voneinander unabhängige Texte effektiver auszuwerten, da durch dieses Embedding jeder einzelne Token aus einem Text von zwei Texten diesem auch mathematisch zugeordnet werden kann.

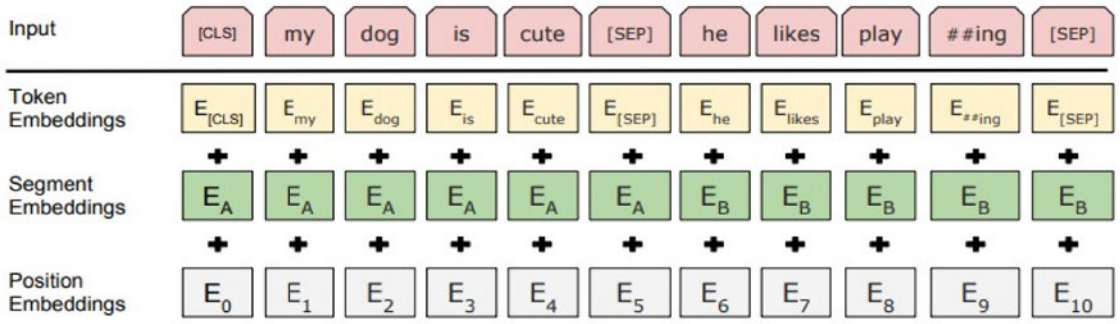

Abbildung 4.1: Aufbau der Eingabevektoren für BERT [4]

In der Abbildung 4.1 ist ersichtlich, dass jeder Token jeweils ein Token Embedding, Segment Embedding und Position Embedding erhält. Es gibt zwei verschiedene Segment Embeddings, E_A und E_B, welche jeweils für einen Text stehen. Die Position Embeddings E_i, wo i eine natürliche Zahl ist, werden für jede Position generiert.

4.1.1 WordPiece Tokenization

Das Problem der unbekannten Tokens wird durch WordPiece Tokenization in der Vorverarbeitung teils gelöst.

Dabei handelt es sich um eine Methode, welche versucht Wörter die häufig vorkommen als Tokens in das Wörterbuch aufzunehmen und Wörter, die selten vorkommen, in Teilwörter zu zerlegen, welche in anderen Wörtern vorkommen und diese ebenfalls in das Wörterbuch aufzunehmen. Wenn beispielsweise das Wort "Bundesausbildungsförderungsgesetz" vorliegt, wird dieses in die Teilwörter/Tokens "Bundes" "ausbildungs" "förderungs" "gesetz" zerlegt, wodurch das Modell die Möglichkeit hat die Bedeutungen aller Tokens zu kombinieren anstatt das gesamte Wort einem Token Embedding zuzuordnen, welches den unbekannten Wörtern entspringt und somit keine Bedeutung hätte.

Der Nutzen ist ebenfalls, dass Eigennamen wie beispielsweise "Cremissimo" aus den Teilwörtern "Crem" und "issimo" zusammengebaut werden können. Wenn die Teilwörter des Eigennamens eine gewisse Beziehung von der Bedeutung her zum Eigennamen selbst haben, dann hat das Sprachmodell die Möglichkeit diesen Eigennamen besser zu verarbeiten, da es die Bedeutung der Teilwörter kennt. Speziell in der deutschen Sprache existiert ein Vorteil durch diese Handhabung, da Wörter zuweilen aus mehreren Wörtern logisch zusammengesetzt werden. [21] [16]

Der Algorithmus der Generierung des Wörterbuches funktioniert wie folgt:

1. Initialisiere das Wörterbuch mit grundlegenden Unicode Ziffern und füge diesem noch ASCII Ziffern hinzu
2. Generiere ein neues Wort bestehend aus zwei im Wörterbuch vorhandenen Einträgen und füge dieses dem Wörterbuch hinzu. Das neue Wort muss maximal häufig, im Gegensatz zu anderen Kombinationen von zwei Wörtern aus dem Wörterbuch, im Datensatz vorkommen
3. Gehe so oft zu Schritt 2. bis eine vordefinierte Größe des Wörterbuches erreicht ist.

Im Nachkommenden wird der Algorithmus durch einen Trainingsdatensatz bestehend aus einem Satz skizziert:

”Mein Nachname ist Schlesig”

Der Satz wird in der Praxis manchmal so abgeändert, dass keine großen Buchstaben mehr vorkommen, um den Algorithmus effektiver einsetzen zu können.

”mein nachname ist schlesig”

Zunächst erhalten wir ein Wörterbuch bestehend aus für die Zielsprache grundlegenden Ziffern. (ASCII/UTF-8/UTF-16 etc.) Das erste Wort, welches hinzugefügt werden würde, ist ”##ch”.

Also ist das Wörterbuch $D = \{\#\#ch\} \cup \{GrundlegendeZiffern\}$

Wenn ein neues Wort ein Wortteil eines anderen Wortes ist, wird dies durch zwei ”##” symbolisiert.

Daraufhin wird das Wort ”##na” gebildet, da dieses ebenfalls zweimal vorkommt. Also ist D = {##na} ∪ {##ch} ∪ {Grundlegende Ziffern}

Weiter werden die einzelnen Wörter solange hinzugefügt, bis die maximale Anzahl von Einträgen erreicht ist.

Wenn wir nun einen Satz vorverarbeiten wird dieser aus Tokens, welche im Wörterbuch D vorhanden sind und durch obigen Algorithmus generiert wurden, durch einen Greedy-Algorithmus zusammengesetzt.

Somit wird aus dem Satz

”mein nachname ist schlesig”

beispielsweise die Folge

”mein”, "nach”, ”##name”, ”ist”, ”schl”, ”##esig”

Daraus ersichtlich ist, dass der Satz von unten nach oben durch Tokens aus D generiert wird und häufig vorkommende Tokens wie ”mein” oder ”ist” in D enthalten sind.

4.1.2 Token Embedding

Die Verwendung von Token Embeddings ist seit Jahren eine gängige Praxis in der Computerlinguistik. Es sind initial zufällig generierte Vektoren, welche jeweils einer natürlichen Zahl 1-x zugeordnet sind und Tokens in mathematischer Form repräsentieren. Jede dieser natürlichen Zahlen von 1-x, wo x die Größe des Wörterbuches ist, hat genau einen korrespondierenden Token im Wörterbuch. Somit hat transitiv betrachtet jeder Token genau ein Token Embedding in Form eines Vektors.
Bijektive Abbildung:

$$t : D \to V, \tag{4.1}$$

V ist Teilmenge von $\mathbb{R}^d$
d ist Element der natürlichen Zahlen und vordefiniert
D = {Alle Tokens die durch die Wordpice Tokenization generiert wurden}

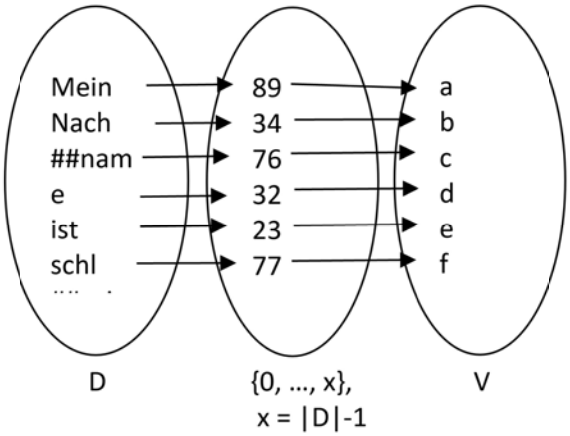

Abbildung 4.2: Darstellung der Transitivität vom Token zum Vektor

Diese Vektoren sind als Repräsentation notwendig damit die Möglichkeit besteht mathematische Operationen auf Tokens durchzuführen. Ebenfalls werden während des Trainings des Modells diese Vektoren ggf. mittrainiert und sind somit Teil des Modells.

Die vorverarbeitete Folge ”mein”, ”nach”, ”##name”, ”ist”, ”schl”, ”##esig” von Tokens wird durch eine durch das Wörterbuch D vordefinierte Folge von natürlichen Zahlen ersetzt: 89, 34, 76, 32, 23, 77

Diese Zahlen werden daraufhin jeweils einem eindeutigen Vektor der Länge d zugeordnet, welche ggf. im Trainingsprozess mittrainiert werden.

Hervorzuheben ist, dass im Kontext von BERT zwischen Token Embeddings und Word Embeddings Unterschiede bestehen. Token Embeddings erhalten nur den Kontext von Tokens, welche in Texten des Trainingsdatensatzes vorzufinden sind, sofern die Token Embeddings mittrainiert werden. Word Embeddings hingegen sind im Rahmen von BERT der jeweils zu dem Token korrespondierende Output des Sprachmodells, was bei der Betrachtung der Architektur klar wird. Dies ermöglicht es auch nach dem Training den Token im Kontext des Eingabetextes widerzuspiegeln, selbst wenn es keinen vergleichbaren Text im Trainingsdatensatz gibt.

4.1.3 Position Embedding

Bei rekurrenten Netzwerken würden die zu den Zahlen 89, 34, 76, 32, 23, 77 korrespondierenden Vektoren sequenziell an das Netz gegeben werden. Dies ist bei BERT jedoch nicht der Fall, da dort alle Vektoren auf Einmal benötigt werden. Das sequentielle Berechnen der Folgen wie bei rekurrenten Netzwerken gibt implizit Informationen darüber an, welche Position die jeweiligen Tokens im Text besitzen. Da in Sprachen die Reihenfolge von Tokens eine semantische und syntaktisch wertvolle Information ist, muss diese auch in das BERT Modell einfließen. Um dieses Problem zu lösen wurden bijektive Abbildungen geschaffen, welche jeder Position bis zu einer maximalen Länge genau einen Vektor zuordnen. Es ist die identische Methode wie bei den Token Embeddings, doch nun ist die maximale Anzahl der vorgenerierten Vektoren festdefiniert und nicht abhängig von der Größe des Wörterbuchs, was bedeutet, dass das Modell nur Texte verarbeiten kann dessen Anzahl an Tokens nach der Vorverarbeitung eine festgelegte Größe nicht überschreitet.

$$p : P \to A, \tag{4.2}$$

A ist Teilmenge von $\mathbb{R}^d$
d ist Element der natürlichen Zahlen und vordefiniert
P = {0, . . . ,511}
Die maximale Anzahl an Position Embeddings ist vom längsten Satz im Trainingsdatensatz abhängig, da das Sprachmodell mit keinen längeren Sätzen trainiert wird, würde es auch keinen Sinn machen mehr Position Embeddings zu erzeugen. Bei BERT ist die maximale Anzahl 512. Ein weiterer Grund für eine Begrenzte Anzahl von Position Embeddings ist die Anzahl der Parameter des Sprachmodells zu begrenzen. Die zu den Token Embeddings t(mein), t(nach), t(##name), t(ist), t(schl), t(##esig) korrespondierenden Position Embeddings sind p(0), p(1), p(2), p(3), p(4), p(5). Dies ist so, da jeder Token genau einer Position im Text zugeordnet werden kann.

4.1.4 Segment Embedding

Das Ziel der Segment Embeddings ist es zwei Folgen von Tokens dem Modell gleichzeitig zur Auswertung geben zu können. Dies ist notwendig, da eines der beiden Trainingsschritte, welcher in Abschnitt 4.7 genauer erläutert werden, dies voraussetzt. Die Embeddings können abermals mittrainiert werden. Für die erste und die zweite Folge von Tokens existiert jeweils genau ein einzigartiger Vektor, wie in der Folgenden bijektiven Abbildung ersichtlich:

$$s : S \to W, \tag{4.3}$$

W ist Teilmenge von $\mathbb{R}^d$
d ist Element der natürlichen Zahlen und vordefiniert
S = {0, 1}
Die 0 und die 1 in S bestimmen ob der Vektor für den ersten Text oder den zweiten Text ist.

Für das folgende Beispiel wurde eine zweite Folge von Tokens definiert als "Das" "ist" "ein" "Te" "##xt".

Um dem Modell noch weitere Informationen über das Ende einer Folge von Tokens zur Verfügung zu stellen wurde ein [SEP]-Token eingeführt. Dieser spezielle Token wird am Ende jeder Folge hinzugefügt. Somit würden den beiden Folgen "mein nach ##name ist schl ##esig" und "Das ist ein Te ##xt" am Ende ein "[SEP]" hinzugefügt.

Tokens	Token Embeddings	Position Embeddings	Segment Embeddings
mein	t(mein)	p(0)	s(0)
nach	t(nach)	p(1)	s(0)
##name	t(##name)	p(2)	s(0)
ist	t(ist)	p(3)	s(0)
schl	t(schl)	p(4)	s(0)
##esig	t(##esig)	p(5)	s(0)
Das	t(Das)	p(6)	s(1)
Ist	t(Ist)	p(7)	s(1)
ein	t(ein)	p(8)	s(1)
Te	t(Te)	p(9)	s(1)
##xt	t(##xt)	p(10)	s(1)

Abbildung 4.3: Beispiel für Segment Embeddings

Text Embeddings werden, im Gegensatz zu Word Embeddings, dazu verwendet eine Vektor Repräsentation des gesamten Textes zu erhalten statt nur eines Tokens im Kontext des Textes.
Da BERT für jeden Eingabevektor einen Ausgabevektor besitzt wird ein [CLS]-Token am Anfang jeder Eingabe hinzugefügt. Die korrespondierende Ausgabe dieses Tokens wird als Text-Embedding verwendet.

Somit würde die endgültige Eingabe des Textes in der Praxis wie folgt aussehen: ”[CLS] mein nach ##name ist schl ##esig [SEP] Das ist ein Te ##xt [SEP]” Bei Betrachtung der Eingabe von zwei Texten kann bereits der erste Bogen von BERT zum BERT Textsplitter geschlagen werden. Der Algorithmus generiert die verschiedenen Eingaben und klassifiziert das Text-Embedding. Die Klassifizierung gibt die Wahrscheinlichkeit an inwieweit zwei Texte aufeinander folgen.

4.2 Zusammenfassung Modelleingabe

Es werden ein oder zwei Sätze mit dem Wordpiece Tokenizer vorab verarbeitet, um die Anzahl an unbekannten Tokens zu reduzieren und es werden die speziellen Token "[SEP]" und "[CLS]" korrekt hinzugefügt. Im zweiten Schritt werden die Token Embeddings berechnet, welche für jeden Token einen einzigartigen Vektor bereitstellen und parallel dazu für jede Position das Position Embedding, welches für jede Position im Text einen einzigartigen Vektor für jeden Token bereitstellt. Des Weiteren wird für jeden Token ein Segment Embedding berechnet. Dies ist ein Vektor, welcher aussagt ob der jeweilige Token zum ersten oder zum zweiten Text gehört. Diese 3 Folgen von Vektoren, welche alle die gleiche Dimension d haben, werden nun für jeden Token jeweils addiert, um den endgültigen Input-Vektor pro Token für das Modell zu erhalten. Das Resultat ist eine Folge von Vektoren der Dimension d. Diese Folge von Vektoren kann nun als Matrix M^{ixd} dargestellt werden, wo i die Länge der Folge ist und d die festgelegte Dimension der Vektoren. Jede Zeile ist somit ein Input-Vektor pro Token.

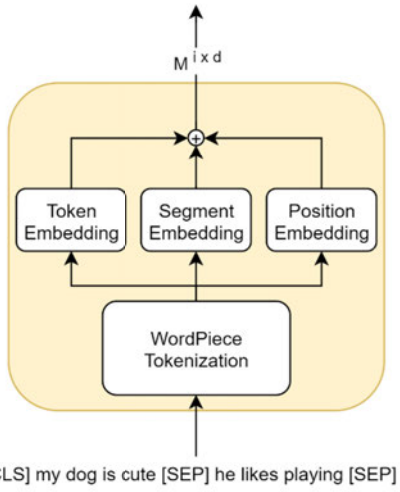

Abbildung 4.4: Ablauf der Eingabegenerierung

$$M = \begin{pmatrix} a_{11} & \cdots & a_{1d} \\ \vdots & \ddots & \vdots \\ a_{i1} & \cdots & a_{id} \end{pmatrix} \tag{4.4}$$

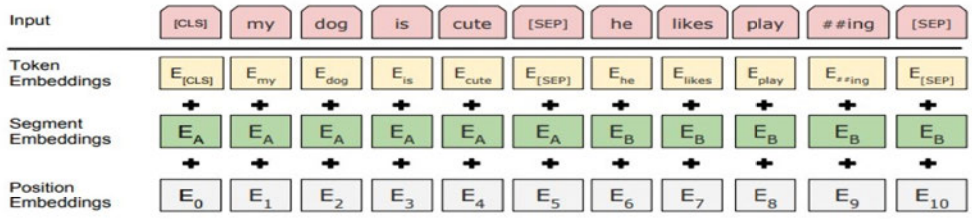

Abbildung 4.5: Aufbau der Eingabevektoren für BERT [4]

4.3 Architektur

Die Architektur von BERT basiert auf einem "Multi-layer bidirection Transformer Encoder".[4] Die Transformer Architektur wurde 2017 im Paper "Attention is all you need" vorgestellt und ermöglicht es ohne Rekurrenz oder Faltung außergewöhnlich effektive Modelle der Computerlinguistik zu trainieren.[18]
Des Weiteren ist es möglich Bilder oder Musik zu generieren. Erwähnenswert ist ebenfalls, dass diese Art der Architektur es möglich macht problemlos Abhängigkeiten in langen Texten einzufangen und ebenfalls viele auswertbare Wahrscheinlichkeiten zwischen Tokens sich anzutrainieren. Auf der Abbildung 4.6 wird die Wahrscheinlichkeit durch die Dicke der Verbindung zwischen jeweils zwei Tokens des gleichen Textes dargestellt.

Diese Wahrscheinlichkeiten könnten als Beziehungen zwischen Tokens interpretiert werden, welche davon abhängig sind wie häufig Tokens gemeinsam in Texten des Trainingsdatensatzes vorkommen, sowie welche Tragweite die Tokens für die jeweiligen Trainingsaufgaben besitzen. Die Beziehungen der Tokens, welche in der Abbildung 4.6 eine starke Bindung zu dem Token "missing"besitzen, scheinen intuitiv betrachtet korrekt zu sein:

1. "this is missing"
2. "what is missing"
3. "what are we missing"

Dies alles ermöglicht die mit dem Transformer neu eingeführte „self-attention“, welche in diesem Kapitel ebenfalls genauer dargelegt wird und erlaubt Texte mit sich selbst in Beziehung zu setzen.

Abbildung 4.6: Darstellung der enkodierten Beziehungen in Transformer-Modellen [18]

4.3.1 Multi-layer bidirectional Transformer encoder

Wie bei der Modelleingabe bereits genauer beleuchtet, dient dem encoder eine Matrix M^{ixd}, wo i die Anzahl an Tokens und d die festgelegte Dimension ist, als Eingabe. Bei genauerer Betrachtung der Abbildung 4.7 stellt man fest, dass die BERT Architektur aus 3 Teilen besteht.

1. Der erste Teil ist die Eingabe, welche einen Text entgegennimmt und eine Matrix M^{ixd} ausgibt.

2. Der zweite Teil der Encoder, welcher die Ausgabe der Eingabeverarbeitung entgegennimmt und eine Ausgabematrix O^{ixd} erzeugt. In dieser Ausgabematrix O sind je nach Trainingsaufgabe beispielsweise in jeder Zeile Embeddings für die einzelnen Tokens vorzufinden, welche den Token im Kontext des Eingabetextes widerspiegeln.

3. Letztlich folgt der Klassifikator, welcher dafür sorgt, dass für die Ausgabematrix O durch beispielsweise einen Linearen Layer einer Klasse zugeordnet wird.

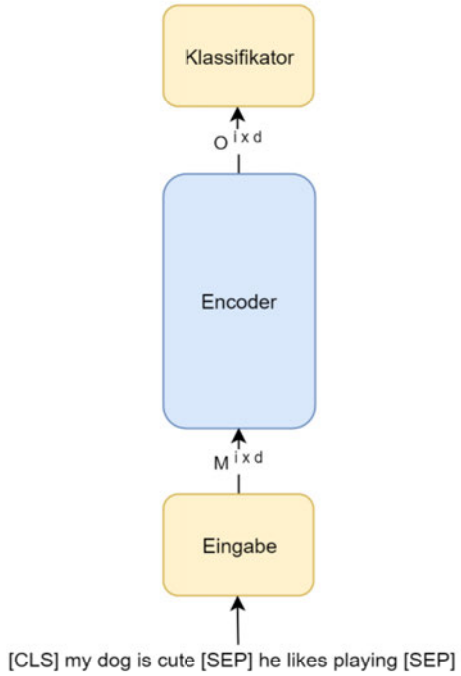

Abbildung 4.7: Black-Box Architektur

Wäre nun das Ziel den gesamten Eingabetext einer Klasse zuzuordnen, so müsste die erste Zeile der Matrix O, da diese der korrespondierende Output der [CLS]-Tokens ist, in einen Linearen Layer gegeben werden, welcher diesen einer Klasse zuordnet.
Der Lineare Layer kann bei 3 Klassen wie folgt verstanden werden:

$$a : \mathbb{R}^d \rightarrow \mathbb{R}^3 \tag{4.5}$$

Es wird meist die Softmax-Funktion auf den Ausgabevektor von a angewendet, um eine Wahrscheinlichkeitsverteilung über die 3 Klassen zu erhalten.[17] Die Klasse mit der höchsten Wahrscheinlichkeit wird für den gesamten Eingabetext gültig gemacht.

So würde bei der Matrix

$$O = \begin{pmatrix} a_{11} & \cdots & a_{1d} \\ \vdots & \ddots & \vdots \\ a_{i1} & \cdots & a_{id} \end{pmatrix} \tag{4.6}$$

der Vektor $\begin{pmatrix} a_1 & \cdots & a_d \end{pmatrix}$ aus der ersten Zeile genommen und in einen Linearen Layer gegeben werden, welcher einen neuen Vektor $V = \begin{pmatrix} b_1 & \cdots & b_3 \end{pmatrix}$ erzeugt. Jeder Eintrag im Vektor V würde für eine Klasse stehen. Im nächsten Schritt wird die Softmax-Funktion auf den Vektor V angewendet, um einer Wahrscheinlichkeitsverteilung über alle Klasse zu erhalten. [17]

Die Softmax-Funktion ist wie folgt definiert:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{4.7}$$

für $i = 1, \ldots, K$ und $V = b_1, \ldots, b_K$ [17]

Intuitiv umschrieben erhält jeder Eintrag des Vektors V eine Wahrscheinlichkeit zugeordnet im Verhältnis zu allen Einträgen des Vektors und die Summe aller Wahrscheinlichkeiten ist eins. Hervorzuheben ist, dass mit steigender Größe ins Negative oder Positive der einzelnen Einträge, die pro Eintrag korrespondierenden mit der Softmax-Funktion generierten Wahrscheinlichkeiten, immer weniger feingranular sind.
Wenn der Eingabevektor beispielsweise wie folgt aussieht V=(42, -24, -10, 89, -79), dann ist $\sigma(V) = (3.87e-21, 8.41e-50, 1.01e-43, 1.00e+00, 1.09e-73)$.
Zu sehen ist, dass die Wahrscheinlichkeiten so gering sind, dass sie an Abstufungen verlieren. Ursprung dieses Problems ist die Natur der Exponentialfunktion. Die Lösung ist eine Skalierung des Vektors. Bei einem Eingabevektor von $V = (0.48, 0.44, 0.05, -0.26, 0.86)$ ist die Ausgabe $\sigma(V) = (0.22, 0.21, 0.14, 0.1, 0.32)$.
Da die Werte des Vektors V gering genug sind, ergeben sich dadurch weniger Probleme. Wichtig zu erwähnen ist, dass der Klassifikator durch anderweitige ersetzt werden kann. Dies ist eine notwendige Bedingung für das Transferlernen, da die Anzahl der Klassen je Trainingsaufgabe variiert und somit die Abbildung dementsprechend angepasst werden muss.
Es ist auch möglich andere Zeilen der Ausgabematrix O des Encoders zu klassifizieren, was im Abschnitt 4.5, welcher sich mit dem Training von BERT befasst, Verwendung findet.

4.3.2 Layer

Der Multi-layer bidirectional Transformer Encoder besteht aus N in Reihe geschalteten Layern, welche jeweils im Aufbau absolut identisch sind.

Dies impliziert, dass für jeden Layer die Eingabematrix genauso groß wie die Ausgabematrix sein muss.

Die Anzahl an Layern zeigt auf, dass die Architektur von BERT auf "deep learning" beruht und somit auch mit den gleichen Problemen von solch tiefen Netzwerken umgehen muss. Ein Problem ist beispielsweise das sogenannte "vanishing gradient problem", welches meist bei sehr tiefen Netzwerken auftritt.[8] Es umschreibt die Problematik, dass mit der Tiefe eines Netzwerkes es immer schwerer wird die ersten Layer genügend anzupassen, da bei beim Gradientenverfahren, dem Zurückpropagieren durch den Berechnungsgraph und das Aktualisieren der Gewichte, die Anpassung dieser immer kleiner wird oder gar nicht mehr stattfindet.
Ferner erinnert der Aufbau vielleicht an Modelle aus der computerunterstützten Bildverarbeitung, wie beispielsweise ResNet.[6]
In der Abbildung 4.8 sind 12 Layer in Reihe geschaltet und es wäre in diesem Fall nun schwer die ersten Layer anzupassen.
Vortrainierte BERT Sprachmodelle gibt es mit unterschiedlichen Anzahlen von Layern. Zumeist sind es 12 oder 24 Layer. Mit der Anzahl an Layern erhöht sich die größe des Berechnungsgraphen des Modells, was es schwerer macht dieses zu trainieren und auch die Voraussetzungen wie leistungsfähige GPUs/TPUs zu erfüllen.

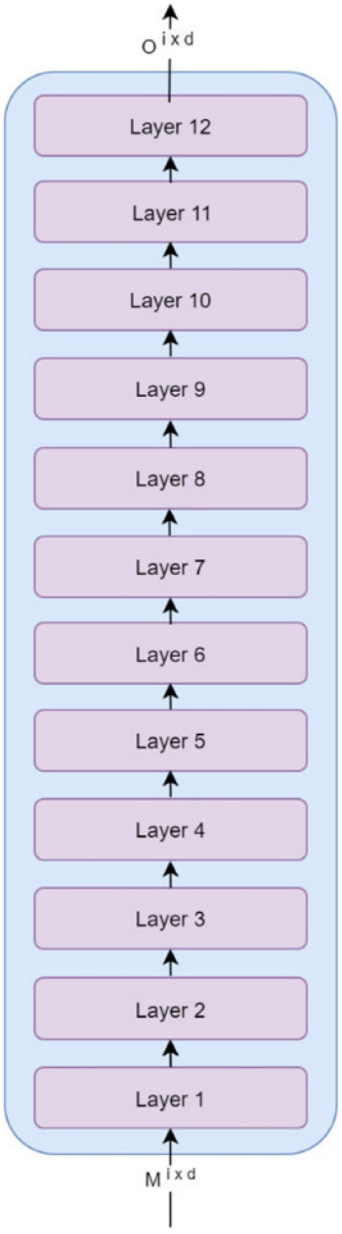

Abbildung 4.8: Encoder mit 12 Layern

Es kann angenommen werden, dass spätere Layer, wie die Layer 8-12 in der Abbildung 4.8, zumeist Beziehungen enthalten, welche für die Trainingsaufgabe wichtig sind. Frühere Layer hingegen sind eher generalisiert und enthalten generelleres Wissen. Da die Architektur und das Training von BERT noch verhältnismäßig neu ist, kann dazu noch nicht allzu genaues gesagt werden. Im Paper "Language Models as Knowledge Bases?" wird bereits so weit gegangen, dass BERT nach dem ersten der zwei Trainings im Transferlernen, in manchen Fällen Fragen beantworten kann, ohne auf diese spezifisch Trainiert worden zu sein.[12] Wie bereits erwähnt sind alle Layer vom Aufbau her identisch. Betrachtet man nun die Layer im Detail sieht man zwei residuale Blöcke. Diese residualen Blöcke sind in der Abbildung 4.9 daran zu erkennen, dass die Ausgabe der Sublayer Feed Forward und Multi-Head Attention mit der Eingabe der jeweiligen Sublayer summiert werden wie auf der Abbildung 4.9 ebenfalls ersichtlich. Dieses Vorgehen ist eine Gegenmaßnahme zum "vanishing gradient problem".[6][8]

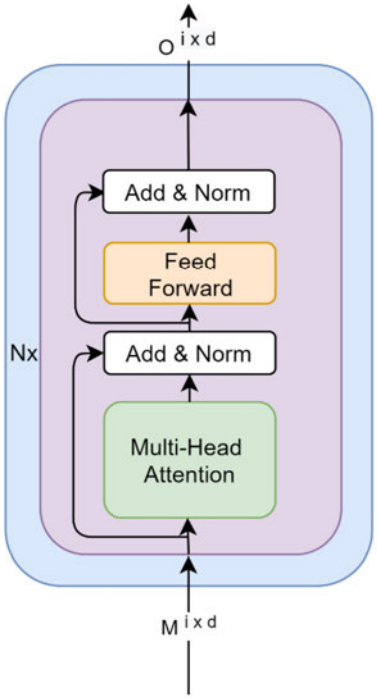

Abbildung 4.9: Innenleben eines Layers

Hervorzuheben ist ein weiteres Mal, dass bei jedem Sublayer die Dimensionen der Eingabe gleich der Ausgabe sind. Die Multi-Head Attention ist der einzige Sublayer in der gesamten Architektur, welcher dazu dient die einzelnen Tokens in Beziehung zu setzen und somit vermutlich der Ursprung der Leistungsfähigkeit ist.
Die Ausgabe der Multi-Head Attention wird in einen Add & Norm Sublayer weitergegeben und daraufhin folgt der zweite residuale Block, welcher aus einem herkömmlichen Feed Forward sowie einem weiteren Add & Norm besteht.[6]

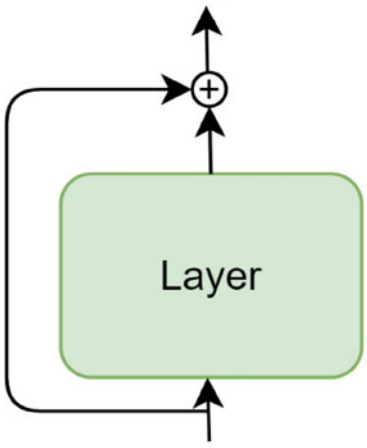

Abbildung 4.10: Residualer Block

4.3.3 Multi-Head-Attention

Im Abschnitt 4.3 zeigte die Abbildung 4.6 auf, wie die Beziehungen zwischen den einzelnen Tokens eines Textes untereinander aussehen. Die Multi-Head-Attention umschreibt den Teil der Architektur, welcher dafür zuständig ist diese Art des in Beziehung setzen von Tokens h mal parallel durchzuführen. Dadurch, dass jeder Layer h Möglichkeiten besitzt sich Beziehungen zwischen den Tokens anzutrainieren, wird das Sprachmodell allumfassender. Die Mächtigkeit des Sprachmodells ist also unter anderem mit der Anzahl an heads in verbinden zu setzen. Die Multihead-attention erhält als Eingabe eine Matrix und erzeugt eine von der Größe her identische Matrix.

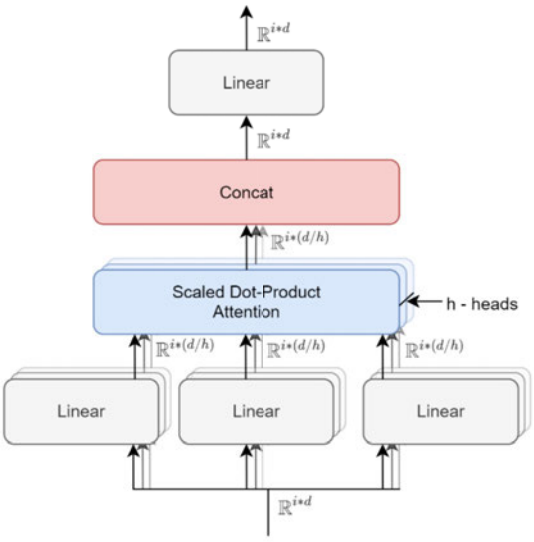

Abbildung 4.11: Berechnungsbaum von h heads

Die parallele Abfolge von Layern erfolgt immer nach demselben Muster.

Die Eingabe Matrix wird für die Erzeugung jedes Heads dreimal in einen jeweils einzigartigen Linearen Layer gegeben, welcher die Dimension d der Eingabematrix auf (d/h) verringert. Daraufhin werden die Ausgaben dieser Layer in die Scaled Dot-Product Attention weitergegeben. Die Ausgabe der h mal parallel ausgeführten Aktionen werden zu einer Matrix wieder zusammengefasst, um eine Matrix der gleichen Dimension der Eingabematrix zu erhalten und ein weiterer Linearen Layer folgt.

Die Verringerung der Dimension der Eingabematrix auf (d/h) ist notwendig, um weiterhin ein halbwegs performantes Modell zu erhalten, da die folgenden Operationen schließlich h mal parallel ausgeführt werden.

Bei einer Eingabematrix $D = \begin{pmatrix} a_{11} & \cdots & a_{1d} \\ \vdots & \ddots & \vdots \\ a_{i1} & \cdots & a_{id} \end{pmatrix}$

würden 3 unterschiedliche Matrizen durch die jeweilige Eingabe in die Linearen Layer entstehen.

$$Q = \begin{pmatrix} a_{11} & \cdots & a_{1(d/h)} \\ \vdots & \ddots & \vdots \\ a_{i1} & \cdots & a_{i(d/h)} \end{pmatrix} \tag{4.8}$$

$$K = \begin{pmatrix} a_{11} & \cdots & a_{1(d/h)} \\ \vdots & \ddots & \vdots \\ a_{i1} & \cdots & a_{i(d/h)} \end{pmatrix} \tag{4.9}$$

$$V = \begin{pmatrix} a_{11} & \cdots & a_{1(d/h)} \\ \vdots & \ddots & \vdots \\ a_{i1} & \cdots & a_{i(d/h)} \end{pmatrix} \tag{4.10}$$

Die Scaled Dot-Product Attention würde diese 3 Eingaben jeweils kombinieren und die Beziehungen errechnen. Die Ausgabe wäre für jeden head eine Matrix

$$head = \begin{pmatrix} a_{11} & \cdots & a_{1(d/h)} \\ \vdots & \ddots & \vdots \\ a_{i1} & \cdots & a_{i(d/h)} \end{pmatrix} \tag{4.11}$$

Wenn angenommen wird, dass 4 heads verwendet werden, so würde 4-mal die Scaled Dot-Product Attention ausgeführt werden und die 4 resultierenden Matrizen würden zu einer großen Matrix zusammengefasst werden, die die Dimensionen der Eingabematrix besitzt.
Darauf folgt noch ein Linearer Layer.

$$MultiHead\,(Q, K, V) = Concat\,(head1, \ldots, head_h)\,W^O \tag{4.12}$$

wo

$$head_i Attention = \left(QW_i^Q, KW_i^Q, VW_i^Q\right) \tag{4.13}$$

[18]

4.3.4 Scaled Dot-Product Attention

Der Name self-attention entspringt der Tatsache, dass ein Text mit sich selbst verglichen wird, wofür das Scaled Dot-Product der Schlüssel ist. Ebenfalls ist es möglich nach den Softmax Wahrscheinlichkeiten auszulesen, welche die Beziehungen zwischen Tokens symbolisieren im Kontext der Trainingsaufgabe.

Q, K und V stehen dabei für Query, Key und Value. Die ursprüngliche Intention dieser Benamung entspringt des Gedankens für eine Anfrage (Query) ein Key-Value Paar zu finden. Intuitiv betrachtet ist dies einen gute Erläuterung, welche nur bedingt ausreichend ist für die Erläuterung der Mächtigkeit dieser Art von Layer in einem „deep learning"-Modell. Zurzeit ist es noch eine offene Forschungsfrage wie genau diese Art von Layer solch einen Vorteil mit sich bringt bzw. wie welches Wissen wann und wo enkodiert wird.

Wie bereits im vorigen Abschnitt ersichtlich, erhält die scaled dot-product attention 3 Matrizen als Eingabe. Bei der Eingabe beginnend wird zunächst die Matrix K transponiert, um eine Matrixmultiplikation zu ermöglichen. Das bedeutet, dass wir nun eine Matrix Q mit den Dimensionen i x (d/h) mit einer Matrix K mit den Dimensionen (d/h) x i multiplizieren. Dies resultiert in dem Produkt der Matrix Q und K, eine Matrix mit den Dimensionen i x i.

Die Zahl i ist nach wie vor die Anzahl der Tokens im Text und daraus schließend ist dies bereits eine Art Vergleich von jedem Token an der Position i mit jedem anderen Token des Textes.

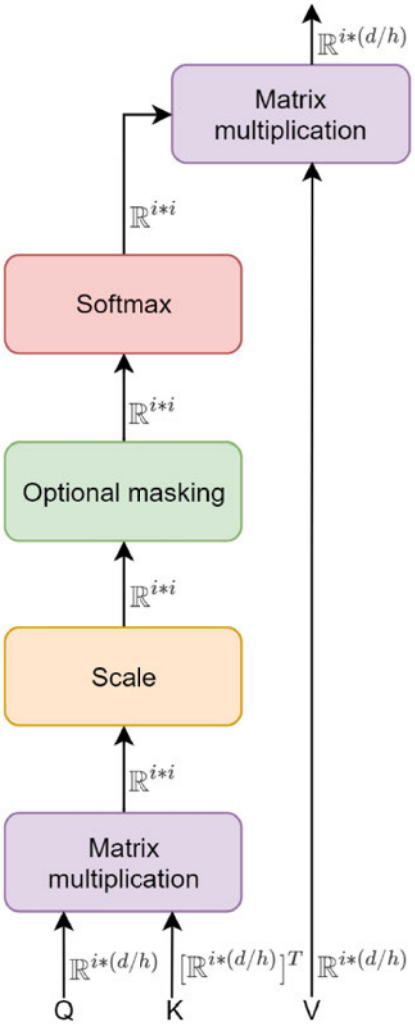

Abbildung 4.12: Reihenfolge der Berechnungen zur scaled Dot-Product Attention

Wie bereits erläutert ist es für die Anwendung der Softmax-Funktion manchmal notwendig Vektoren zu skalieren, was im nächsten Schritt auch getan wird. Diese erfolgt durch das Multiplizieren der Matrix mit: [18]

$$\frac{1}{\sqrt{d}} \tag{4.14}$$

Das Maskieren der Matrix ist notwendig, da in der Praxis mit sogenannten Batches gerechnet wird. Dies bedeutet, dass mehrere Texte gleichzeitig in das Modell gegeben werden, um einen gemeinsamen Fehler zu errechnen. Notwendig für diese Technik ist es jedoch alle Texte auf eine gleiche Länge zu bringe, da Matrixmultiplikationen dies voraussetzen. Dies wird mit einem sogenannten [PAD]-Token vollbracht. Alle Texte, die kürzer sind als der längste Text, werden durch diesen Token auf eine gleiche länge gebracht. Im Vergleich von jedem Token mit jedem anderen soll dieser spezielle Token jedoch ignoriert werden, weswegen die Maskierung erfolgt. Es folgt die Softmax-Funktion, welche auf jede Zeile angewendet wird und somit die Wahrscheinlichkeiten angibt. Zuletzt erfolgt eine Matrixmultiplikation mit V um die Dimensionen der Eingangsmatrizen als Ausgangsmatrix zu erhalten. Formalisiert kann die scaled dot-product attention wie folgt dargestellt werden: [18]

$$Attention\,(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d}}\right)V \tag{4.15}$$

Verbildlicht dargestellt wird die Matrix

$$Q = \begin{pmatrix} a_{11} & \cdots & a_{1(d/h)} \\ \vdots & \ddots & \vdots \\ a_{i1} & \cdots & a_{i(d/h)} \end{pmatrix} \tag{4.16}$$

mit der Matrix

$$K = \begin{pmatrix} a_{11} & \cdots & a_{1i} \\ \vdots & \ddots & \vdots \\ a_{(d/h)1} & \cdots & a_{(d/h)i} \end{pmatrix} \tag{4.17}$$

multipliziert um eine Matrix

$$X = \begin{pmatrix} a_{11} & \cdots & a_{1i} \\ \vdots & \ddots & \vdots \\ a_{i\ 1} & \cdots & a_{i\ i} \end{pmatrix} \tag{4.18}$$

zu erhalten. Diese Matrix X wird daraufhin zur Vorbereitung auf die Softmax-Funktion skaliert:

$$Y = \frac{1}{\sqrt{d}} X \tag{4.19}$$

Die Matrix Y mit den Dimensionen i x i wird nun in eine Softmax-Funktion gegeben wodurch die Beziehungen der Tokens darstellbar werden. Diese werden wie folgt skizziert:

	[CLS]	my	dog	is	cute	[SEP]	he	likes	play	##ing	[SEP]
[CLS]	0.04	0.15	0.08	0.15	0.15	0.03	0.08	0.04	0.14	0.07	0.07
my	0.18	0.06	0.06	0.04	0.09	0.14	0.07	0.12	0.08	0.11	0.05
dog	0.10	0.11	0.04	0.14	0.04	0.05	0.18	0.08	0.14	0.10	0.03
is	0.05	0.08	0.12	0.13	0.06	0.03	0.08	0.09	0.17	0.13	0.06
cute	0.03	0.10	0.04	0.08	0.07	0.17	0.08	0.18	0.03	0.19	0.05
[SEP]	0.08	0.17	0.18	0.04	0.06	0.07	0.17	0.09	0.05	0.06	0.04
he	0.03	0.15	0.06	0.05	0.05	0.21	0.07	0.04	0.14	0.14	0.05
likes	0.07	0.08	0.11	0.05	0.14	0.05	0.04	0.17	0.06	0.04	0.20
play	0.15	0.04	0.11	0.11	0.11	0.15	0.03	0.03	0.02	0.15	0.09
##ing	0.05	0.18	0.04	0.04	0.04	0.19	0.04	0.04	0.07	0.19	0.12
[SEP]	0.20	0.13	0.05	0.07	0.05	0.11	0.11	0.05	0.13	0.04	0.04

Abbildung 4.13: Wahrscheinlichkeiten von Token zu Token

Die Ausgabe wird letztlich noch mit

$$V = \begin{pmatrix} a_{11} & \cdots & a_{1(d/h)} \\ \vdots & \ddots & \vdots \\ a_{i1} & \cdots & a_{i(d/h)} \end{pmatrix} \tag{4.20}$$

multipliziert, um die ursprünglichen Dimensionen der Eingabematrix i * (d/h) zu erhalten.

4.3.5 Add & Norm und Feed Forward

Der Add & Norm Sublayer besteht aus der für einen residualen Block notwendigen Addition und aus einem Norm-Layer. Dieser Norm-Layer besteht aus einem Linearen Layer sowie einen darauffolgenden Dropout.

Feed Forward ist ein Linearer Layer, welcher durch die GELU Aktivierung aktiviert wird. Die GELU Aktivierung scheint im Vergleich zur RELU und ELU Aktivierung bessere Ergebnisse bei gleichbleibender Performance zu erzielen.[7] Während in der Multi-Head Attention die Kombination und das lernen der Beziehungen der einzelnen Tokens im Kontext der Trainingsaufgabe im Vordergrund steht, geht es beim Feed Forward darum die einzelnen Tokens isoliert von den anderen Tokens zu modifizieren.

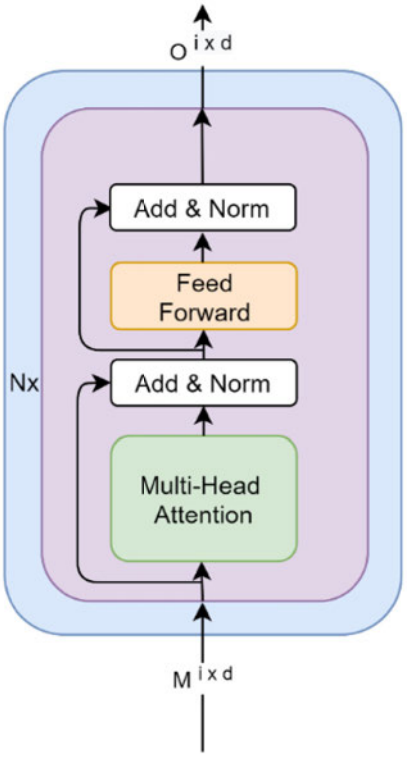

Abbildung 4.14: Innenleben eines Layers

4.4 Architektur Zusammenfassung

Zusammenfassend wird in jeden der N Layer zunächst die Multi-head-attention auf die Eingabe angewendet.

Die in der Multi-head-attention zu findende Scaled Dot-Product Attention ist maßgeblich daran beteiligt die Beziehungen zwischen den Tokens zu ermitteln.

Im nächsten Schritt wird die Eingabe auf die Ausgabe der Multi-head-attention addiert. Diese Addition ist notwendig, um von den Vorteilen der residualen Blöcke zu profitieren.

Eine Normalisierung gegen Übertraining folgt. Der nächste residuale Block besteht aus einem Feed Forward Layer und einer weiteren Normalisierung.

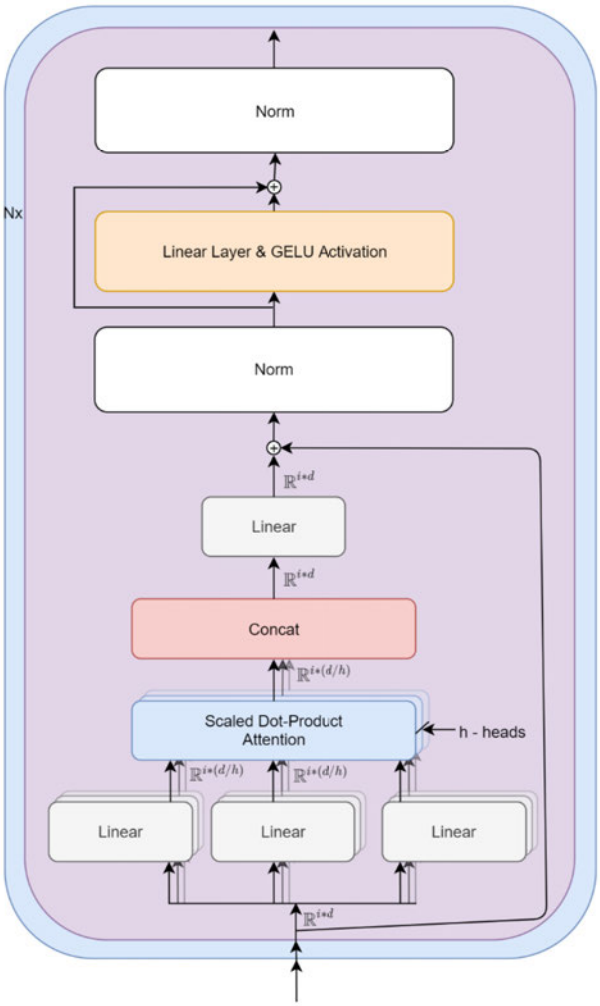

Abbildung 4.15: Gesamte Architektur im Überblick

4.5 Training

Während die grundlegende Architektur von BERT bereits einige Jahre alt ist, kann das Training als innovativ bezeichnet werden. Es ist eine neue Kombination von bestehenden Techniken.

Das Training besteht aus 2 Schritten. Zunächst wird dem Sprachmodell die Zielsprache bzw. die Zielsprachen und die damit einhergehende Syntax und Semantik durch das Masked Language Model-Training beigebracht. Im zweiten Schritt wird durch die Next Sentence Prediction dem Sprachmodel gezeigt wie es grundlegend mit 2 Eingabetexten umgehen kann. Dies ist eine Vorbereitung auf weit verbreitete Benchmarks wie SQUAD 2.0. [14] Bei SQUAD 2.0 ist es das Ziel zu einer Frage eine Antwort unter vielen möglichen Antworten zu finden. Dies ist beispielsweise notwendig, wenn eine Frage einer Webseite zugeordnet werden soll, welche die Antwort besitzt.

4.5.1 Masked Language Model

Das Masked Language Model führt neben dem [SEP]-Token und dem [CLS]-Token einen weiteren speziellen Token ein. Das [MASK]-Token nimmt dem Platz eines oder mehrere Tokens im Eingabetext ein, mit dem Ziel die zu dem Token korrespondierende Ausgabe des Modells darauf zu trainieren den ursprünglichen Token im Text zu klassifizieren. Dies ist vergleichbar mit Skip-gram oder N-gram. Durch dieses Training werden dem Sprachmodel Beziehungen zwischen Tokens antrainiert und somit implizit auch die Syntax und Semantik. Da in weiteren Trainings des Sprachmodells jedoch das [MASK]-Token nicht mehr vorkommt muss es ebenfalls ohne diesen Token trainiert werden. Folgende Eingabedaten werden generiert:

- 15% der Tokens werden für eine Voraussage ausgewählt
- 80% dieser ausgewählten Tokens werden mit dem [MASK]-Token maskiert
- 10% dieser Tokens werden mit einem anderen zufälligen Token ersetzt
- 10% dieser Tokens werden mit dem korrekten Token belassen

Somit werden beispielsweise Texte der Form "[CLS] Mein [MASK] ist Hans [SEP]" als Eingabe verwendet. Daraus ersichtlich ist, dass BERT für die Vorhersage des Tokens "Name", welcher durch den [MASK]-Token ersetzt wurde, sowohl den linken als auch den rechten Text erhält. Aufgrund dessen steht das B in BERT für Bidirectional.

4.5.2 Sentence Prediction

Das Training der Next Sentence Prediction ermöglicht den vorgestellten BERT Textsplitter Texte überhaupt erst zu zerteilen.

Der Corpus auf welchen das Sprachmodell trainiert wurde ist eine Kombination des englischen Wikipedia und des BooksCorpus. [25]

Aus diesem Datensatz werden Satzpaare ausgelesen. 50% der Satzpaare bestehen aus 2 Sätzen, die aufeinanderfolgen und 50% der Satzpaare aus Sätzen, die nicht aufeinanderfolgen. Das Ziel ist es ein Sprachmodell zu erhalten, dass entscheiden kann ob 2 Sätze aufeinanderfolgen oder nicht. Dies impliziert, dass das Sprachmodell mehr oder minder gelernt hat zu verstehen ob zwei Sätze Ähnlichkeiten besitzen bzw. im gleichen Themenbereich angesiedelt sind.

5 Experimente

In diesem Abschnitt wird der alleinstehende Klassifikator mit einem herkömmlichen Textsplitter-Klassifikator, welcher auf regulären Ausdrücken basiert und dem BERT-Textsplitter Algorithmus verglichen. In allen 3 Fällen wird der einmalig trainierte Klassifikator verwendet.

Des Weiteren wird erläutert, wie der verwendete Klassifikator aufgebaut wird, welcher Datensatz verwendet wird sowie einige weitere Formalien.

5.1 Datensatz

Für die Experimente muss ein Datensatz gewählt werden, welcher umgangssprachliche Texte enthält und gleichzeitig viele Klassen besitzt. Eine hohe Anzahl der Klassen ist notwendig, um zu demonstrieren, dass die vom Algorithmus ausgewählte Klasse nicht durch Zufall richtig gewählt wird.

Der opinosis opinion dataset enthält Texte, welche 51 verschiedenen Klassen zugeordnet werden. Durchschnittlich gibt es für jede Klasse annähernd 100 Texte. Die Texte wurden aus verschiedenen Quellen extrahiert, darunter "Tripadvisor" und "Amazon.com". [5] Die Texte bestehen aus englischsprachigen Rezensionen. Jede Rezension wird einer Kategorie bzw. Klasse zugeordnet. Es gibt Kategorien wie beispielsweise "comfort_honda_accord_2008" oder auch "video_ipod_nano_8gb". Hervorzuheben ist, dass manche Klassen aus der menschlichen Intuition heraus sehr nah beieinander liegen und somit auch für den Klassifikator voraussichtlich schwer zu differenzieren sind. Ein Beispiel für nah beieinanderliegende Klassen sind die beiden Kategorien "food_holiday_inn_london" und "food_swissotel_chicago" in welchen die Texte zum Teil sich nur auf die Qualität des Essen beziehen und somit der Klassifikator implizit lernen muss in welchen Hotel das Essen besser oder schlechter schmeckt oder gar inwieweit sich Kunden des jeweiligen Hotels sprachlich ausdrücken. Ein Datenpunkt aus dem einen Datensatz ist "The food for

our event was delicious." und aus dem anderen wiederum "Everything about this hotel was first class including the food!" .

Vor dem Training wurde der Datensatz in 2 Datensätze zerteilt. 75% der Datenpunkte werden für das Training verwendet und 25% für die anschließende Evaluation.

5.2 Technologieplattformen

Das Projekt ist in Python 3.7 geschrieben, da die Verbreitung dieser Programmiersprache im Bereich des "deep learnings" am Größten ist, woraus folgt, dass die neuesten Entdeckungen in dieser Sprache veröffentlicht werden.

Pytorch wurde als "deep learning" Basisframework des Projektes gewählt. Vorteilhaft an Pytorch ist, dass dieses Framework kaum von der Python Syntax abweicht und somit einen einfachen Einstieg für Personen, die sich mit Python auskennen, erlaubt. Gleichzeitig ist es aber auch möglich sehr tiefgehende Anpassungen vorzunehmen. [11]
Flair ist ein auf Pytorch aufbauendes Natural Language Processing Framework, welches den Benutzern erlaubt auf einfachste Weise NLP Modelle zu trainieren. Dieses Framework wurde dazu verwendet den Klassifikator zu trainieren. Des Weiteren wurden einige Klassen dieses Frameworks in der Implementation des Algorithmus verwendet oder erweitert. [1]
Das Transformers Framework, vormals pytorch-pretrained-bert, ist ein auf Pytorch basierendes Framework, welches unzählige vortrainierte BERT Modelle oder auf BERT basierende Modelle in Pytorch zur Verfügung stellt. [20]
Hintergrund ist, dass viele Sprachmodelle in Tensorflow trainiert wurden und dieses Framework die in Tensorflow trainierten Modelle zu Pytorch kompatiblen Modellen umwandeln kann. Mit diesem Framework wurde ein Teil des BERT – Textsplitter realisiert. Mittlerweile ist dieses Framework so verbreitet, dass Facebook-Research deren Sprachmodell RoBERTa, welches eine Weiterentwicklung des ursprünglichen BERT ist und zeitweise "State of the art" war, durch dieses Framework realisiert hat. [9]
Das Natural Language Toolkit wurde verwendet, um den auf regulären Ausdrücken basierenden Textsplitter zu realisieren. [2]

Es wurde Google Colab für die Experimente verwendet, da durch dieses Werkzeug mit geringem Aufwand gegebene Ziele erreicht werden können.

5.3 BERT Modell

Mittlerweile gibt es unzählige vortrainierte BERT Sprachmodelle, welche sich in der Trainingsaufgabe sowie in deren Parametern unterscheiden. Für den BERT-Textsplitter wurde das Sprachmodell "bert-large-cased" verwendet, welches aus 24 Layern und 16 Heads besteht, sowie mit einer Dimension von 1024 verwendet werden kann.
Das gesamte Sprachmodell besteht aus 340-millionen Parametern. Diese Parameter bestehen aus den gespeicherten Token Embeddings, Segment Embeddings, Position Embeddings sowie allen Gewichten der Linearen Layer.
Das "cased" im Namen drückt aus, dass dieses Sprachmodell mit Berücksichtigung der Groß und Kleinschreibung trainiert wurde. [20]

5.4 Klassifikator

Wie bereits erwähnt wird der Klassifikator durch das Framework flair realisiert. Der Klassifikator besteht aus 3 Teilen.

Zunächst generiert BERT für jeden Token ein Word Embedding. Aufgrund der Vorverarbeitung von BERT kann ein Wort in mehrere Tokens zerteilt werden. Es wird der Durchschnitt der korrespondierenden Ausgaben pro Token genommen, um eine Repräsentation für das gesamte Wort zu erhalten.[1]

$$\frac{1}{n}(v_1 + v_2 + \ldots + v_n) \tag{5.1}$$

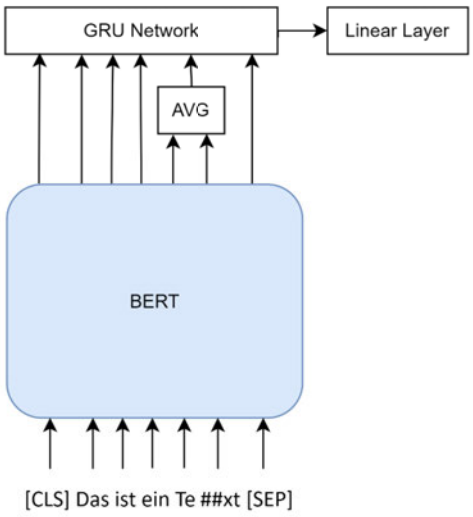

Abbildung 5.1: Aufbau des für die Experimente verwendeten Klassifikators

v_n, wo n eine natürliche Zahl ist, sind dabei die korrespondierenden Ausgaben von BERT für die Tokens, welche gemeinsam ein Wort ergeben. Die Word Embeddings werden folgend in ein Gated Recurrent Neural Network gegeben. Die letzte Ausgabe dieses Netzwerkes repräsentiert dabei den gesamten Text und wird final in einen Linearen Layer gegeben, welcher die Klassifikation ermöglicht. [3]

BERT wird in diesem Training nicht mittrainiert und dient ausschließlich der Generierung von Word Embeddings.

Das für den Klassifikator verwendete vortrainierte Sprachmodell heißt "bert-base-cased", besteht aus 12 Layern, 12 heads und verwendet eine Dimension von 768. Das Model hat 110-millionen Parameter.

Der Lineare Layer verringert die Dimension der Ausgabe des GRU Network auf die Anzahl der Klassen. Die auf die Ausgabe angewendete Softmax-Funktion wird nicht im Modell gespeichert und somit hier auch nicht angegeben.

5.5 Textsplitter basierend auf regulären Ausdrücken

Herkömmlicherweise wird ein Text, sofern die Wahrscheinlichkeit nicht den Schwellwert überschreitet, in mehrere Textteile zerteilt. Diese Zerteilung basiert zumeist auf regulären Ausdrücken. Es wurde zu Vergleichszwecken solch ein Algorithmus implementiert.
Algorithmus:

1. Klassifiziere den Text
   a) Sofern die Wahrscheinlichkeit für die Klasse höher als der Schwellwert ist, gebe diesen mit der Klasse zurück
   b) Falls die Wahrscheinlichkeit geringer als der Schwellwert ist, so gehe zu Punkt 2.
2. Zerteile den Text in Sätze mit Hilfe von regulären Ausdrücken
3. Klassifiziere jeden Satz und gib die Klasse mit der höchsten Wahrscheinlichkeit zurück

5.6 Training

So wird ein Klassifikator auf den opiniosis Datensatz trainiert. Als Fehlerfunktion wird die Kreuzentropie gewählt.[10]

$$-\sum_{x \epsilon X} p\left(x\right) log q\left(x\right) \tag{5.2}$$

p(x) gibt die Zahl zurück, welche für gegebenen Text an gegebenen Vektoreintrag x als Ausgabe des Modells erwartet wird. Diese Zahl ist somit 0, wenn der Vektoreintrag bei gegebener Eingabe 0 sein soll und die zum Vektoreintrag korrespondierende Klasse nicht die richtige ist. Der Vektoreintrag ist 1 sofern dieser die richtige Klasse widerspiegelt. q(x) ist die berechnete Wahrscheinlichkeit des Modelles für jeden Vektoreintrag x.
Wenn der Zielvektor und der Ausgabevektor wie folgt aussehen:

$$Z = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \; und \; P = \begin{pmatrix} 0.1 \\ 0.4 \\ 0.5 \end{pmatrix} \tag{5.3}$$

p(1) würde 0 zurückgeben und q(1) = 0.1.

Zur Optimierung des Modells wurde das ADADELTA-Verfahren gewählt. Dieses Verfahren wendet eine Lernrate auf jede Dimension an und ermöglicht es automatisiert die Hyperparameter zu optimieren. [24] Die Minibatchgröße wurde auf 16 während des Trainings festgelegt. Dies ist eine technische Begrenzung. Minibatches verhindern Übertraining bzw. eine zu starke Spezialisierung des Modells auf die Trainingsdaten. Die initiale Lernrate wurde auf 0.1 gesetzt. Wenn in einer Epoche der Fehler sich nicht mindern lässt, oder gar steigt, wird die Trainingsrate mit 0.05 multipliziert, um eine schnellere Anpassung zu ermöglichen.[6] Es gibt zwei Konvergenzkriterien: Zunächst wird die maximale Anzahl von Epochen auf 1000 festgelegt. Des Weiteren wird das Training beendet, wenn die Lernrate die Konstante 0.0001 unterschreitet. Dies ist notwendig, da ab einem bestimmten Punkt die Anpassungen so minimal werden, dass sich ein weiteres Training nicht mehr rentiert.

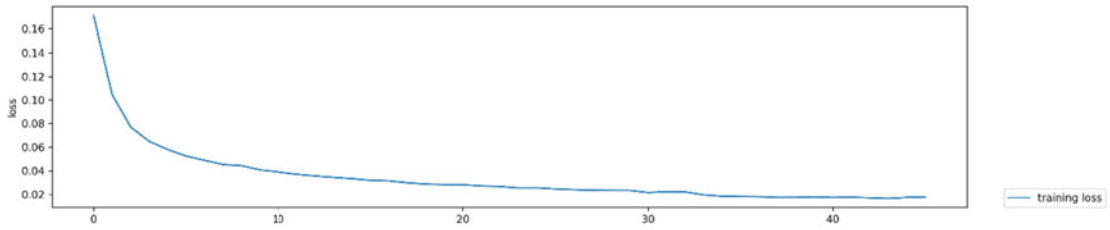

Abbildung 5.2: Fehler über Epochen

5.7 Auswertung

Für die Auswertung wurden True Positive (TP), False Negative (FN) und False Positive (FP) eigens definiert. Dies liegt daran, dass man bei einem Chatbot noch die Möglichkeit hat einen Dialog zu führen und somit andere Bedingungen gelten.

True Positive wird definiert als eine richtige Vorhersage, dessen Wahrscheinlichkeit den Schwellwert überschreitet.

False Positive ist eine falsche Vorhersage, welche den Schwellwert überschreitet.

False Negative sind Vorhersagen, welche den Schwellwert nicht überschreiten.

Dies kommt aus der praktischen Anwendung von Klassifikatoren und stammt nicht aus der allgemeingültigen Definition dieser Namen.

Die Grundlage für diese Definition bildet die Tatsache, dass in einem Dialog der Anwender gefragt werden kann ob die Vorhersage des Modells der Absicht seiner Anfrage entspricht. Durch das Nachfragen hilft der Anwender dabei den Datensatz zu erweitern und somit zukünftige Trainings zu verbessern.

Ziel ist es im Chatbot Anwendungsfall so wenig False Positive wie möglich zu haben. Dies ist damit begründet, da falsche Antworten den Eindruck eines nicht intelligenten Chatbots erwecken, wohingegen Nachfragen ob dies die richtige Antwort ist, dass nicht unbedingt verursachen.

Die allgemeine Definition erhält auch True Negatives. True Negatives werden in dieser Arbeit nicht zur Anwendung gebracht, da diese die "Accuracy" verzehren würden.
Wenn nur eine Vorhersage ein True Positive ist, dann gibt es (Anzahl der Klassen minus eins) True Negatives mehr und bei einem False Positive (Anzahl der Klassen minus zwei) True Negatives. Dieser Wert würde die Auswertung verzerren, da er bei 51 Klassen denkbar hoch sein würde.

Der optimale Schwellwert wird in diesen Abschnitt ermittelt.

Die Precision wird wie folgt definiert:

$$P = \frac{TP}{(TP + FP)} \tag{5.4}$$

Die Precision gibt an wie viele Datenpunkte richtig mit einer schwellwertüberschreitenden Wahrscheinlichkeit klassifiziert werden im Verhältnis zu sich selbst und den fälschlicherweise den Schwellwert überschreitenden Datenpunkten. Für einen Chatbot ist dieser Wert wichtig, da ein hoher Wert zeigt, dass es wenig False Positives gibt.

Die Accuracy ist wie folgt definiert:

$$A = \frac{TP}{(TP + FP + FN)} \tag{5.5}$$

Die Accuracy gibt an inwieweit das Modell im Verhältnis zu allen Datenpunkten abschneidet und somit wie die generelle Performance ist.

Der Recall ist wie folgt definiert

$$R = \frac{TP}{(TP + FN)} \tag{5.6}$$

Der Recall gibt an wie sicher sich das Modell ist.

Der F1 Score ist wie folgt definiert:

$$F1 = 2 * \frac{P * R}{P + R} \tag{5.7}$$

Der F1 Score stellt eine Harmonie zwischen Recall und Precision her.

Im Folgenden werden die Ergebnisse der Experimente dargelegt.

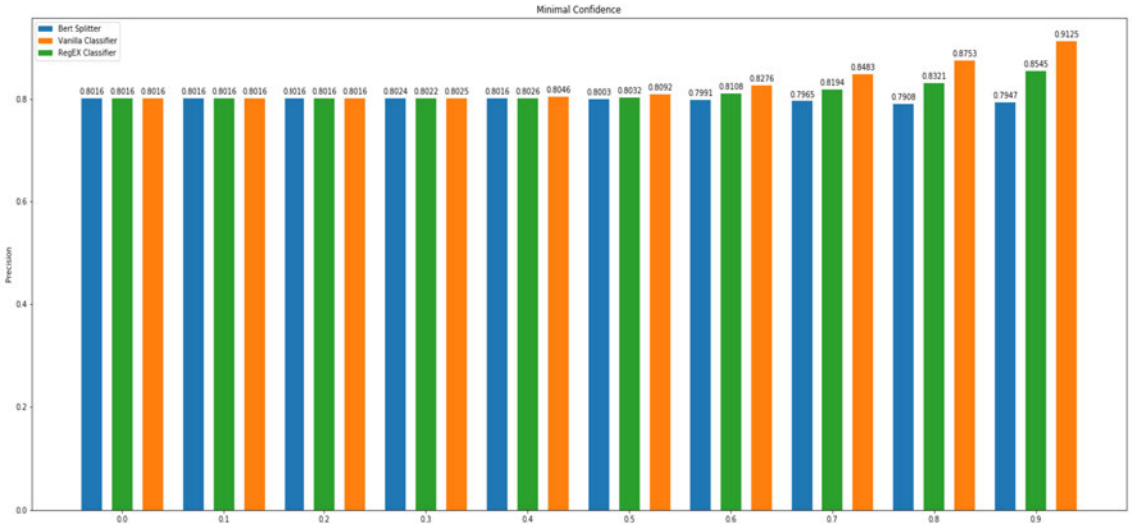

Abbildung 5.3: Precision über Schwellwert. Pro Schwellwert ein Eintrag jeweils für den BERT Textsplitter Algorithmus, alleinstehenden Klassifikator und dem Klassifikator in Kombination mit regulären Ausdrücken

Der BERT Textsplitter-Algorithmus scheint in der Precision mit steigenden minimalen Schwellwert am schlechtesten abzuschneiden. Dahinter folgt der auf regulären Ausdrücken basierende Algorithmus. Am besten schneidet der alleinstehende Klassifikator ab.

Dies liegt an der Definition der Precision. Der BERT-Textsplitter und auf regulären Ausdrücken basierende Textsplitter Algorithmus geben sich nicht mit einer Wahrscheinlichkeit kleiner dem Schwellwert zufrieden und suchen aktiv nach einer Klasse mit einer höheren Wahrscheinlichkeit die den Schwellwert überschreitet. Durch diese Suche kann es aber auch vermehrt dazu kommen, dass sich aus der Unsicherheit des Modells eine Fehlentscheidung bildet. Diese Fehlentscheidung gilt dann als False Positive, was im Gegensatz zu False Negative mit in die Berechnung der Precision einfließt.

Je höher der Schwellwert umso länger suchen diese Algorithmen nach einer Klasse mit einer höheren Wahrscheinlichkeit und desto wahrscheinlicher wird es, dass diese Suche ein False Positive zurückgibt. Dabei ist ein hoher Schwellwert insofern von Vorteil, dass nur eine richtige Antwort als True Positive gesehen wird bzw. vom Chatbot zurückgegeben wird, wenn sich der Klassifikator absolut sicher ist. Ansonsten würde der Anwender gefragt werden um der Klassifikation seiner Absicht entspricht.

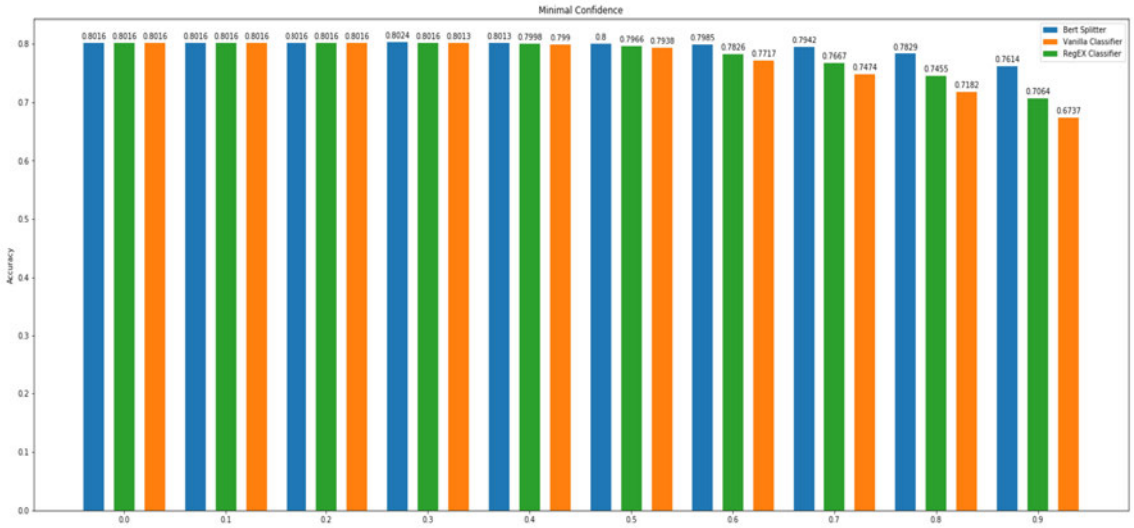

Abbildung 5.4: Accuracy über Schwellwert. Pro Schwellwert ein Eintrag jeweils für den BERT Textsplitter Algorithmus, alleinstehenden Klassifikator und dem Klassifikator in Kombination mit regulären Ausdrücken

Die Accuracy hingegen scheint mit steigendem Schwellwert zu zeigen, dass im Gegensatz zum alleinstehenden Klassifikator die anderen beiden Algorithmen einen Vorteil besitzen. Dies äußert sich in einer durchaus höheren Performance. Die Ursache dafür ist, dass die anderen beiden Algorithmen signifikant mehr True Positive Vorhersagen ermittelt haben als mit dem alleinstehenden Klassifikator.

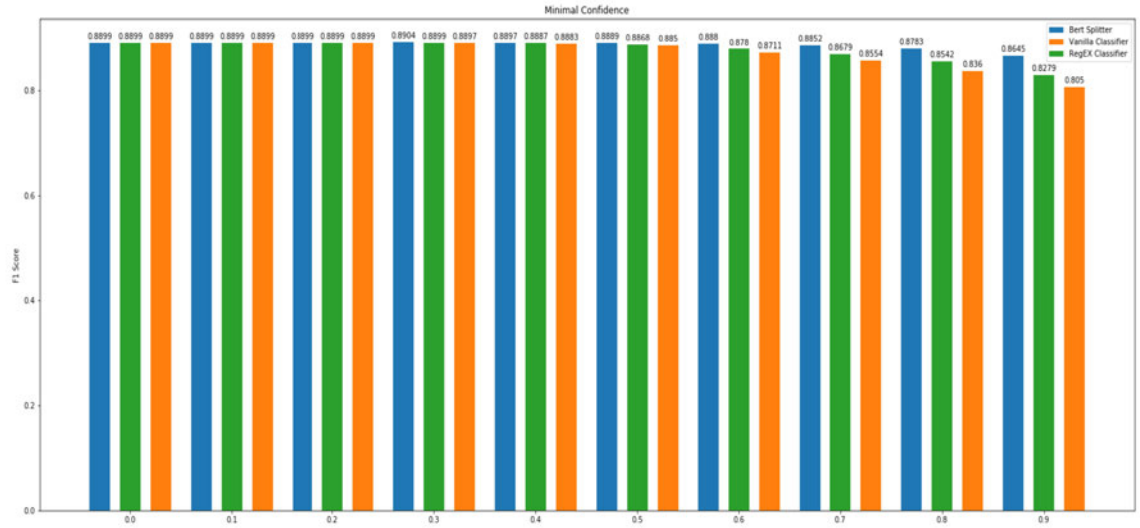

Abbildung 5.5: F1 Score über Schwellwert. Pro Schwellwert ein Eintrag jeweils für den BERT Textsplitter Algorithmus, alleinstehenden Klassifikator und dem Klassifikator in Kombination mit regulären Ausdrücken

Das Gleiche lässt sich im F1 Score betrachten, wo der BERT Textsplitter - Algorithmus mit steigendem Schwellwert am besten abschneidet.

6 Konklusion

Zusammenfassend sind die Ergebnisse nur bedingt nutzbringend. Chatbots benötigen eine möglichst hohe Precision. Dies ist damit zu erklären, dass Chatbots Konversationen führen. In Konversationen ist es möglich bei Unsicherheit Nachfragen zu stellen, um sich selbst zu bestätigen. Dies wäre bei einem False Negative möglich. Der Benutzer könnte gefragt werden ob seine Absicht richtig erkannt wurde. Bei einem False Positive hingegen würde das System selbstsicher eine falsche Antwort geben.
Die Selbstsicherheit eines Systems falsche Antworten als richtig zurückzugeben muss natürlich so gering wie möglich gehalten werden, um intelligent zu wirken.
In Betracht dessen, dass der Klassifikator und ebenfalls der Algorithmus jeweils BERT verwenden ist dies eine extrem Hardwarefordernde Lösung, welche produktiv nur bedingt einsetzbar ist. Die Antwortzeiten eines solchen Systems müssen schließlich auch im Rahmen bleiben. Viele Weiterentwicklungen von BERT gehen dieses Problem bereits an. [15]
Sinnvolle Anwendungsgebiete wären daher Klassifikationsaufgaben, in welchen umgangssprachliche Texte klassifiziert werden und zudem die Klassifikation nicht direkt Kunden zurückgegeben wird oder rechtlich komplex sind, wie die Hassredenerkennung.
Ein Beispiel dafür wäre der verwendete Datensatz, welcher Kundenbewertungen versucht Themengebieten zuzuordnen um somit eine manuelle Zuordnung von Kundenbewertungen überflüssig macht, sofern die Kunden selbst sich nicht zugeordnet haben.
Des Weiteren könnten bessere Ergebnisse erzielt werden, wenn die Daten um eine Klasse erweitert werden würden. Diese Klasse würde nur Text beinhalten, welcher nichts mit den ursprünglichen Themen/Klassen zu tun hat und somit es ermöglichen den BERT Textsplitter-Algorithmus so zu erweitern, dass Textteile, welche dieser Klasse zugeordnet werden würden, ignoriert werden. Beispielsweise könnten Untertitel-Texte verwendet werden, da diese umgangssprachlich sind und mit größerer Wahrscheinlichkeit keine themenspezifischen Texte beinhalten.

Literaturverzeichnis

[1] Akbik, Alan ; Blythe, Duncan ; Vollgraf, Roland: Contextual String Embeddings for Sequence Labeling. In: *COLING 2018, 27th International Conference on Computational Linguistics*, 2018, S. 1638–1649

[2] Bird, Steven ; Klein, Ewan ; Loper, Edward: *Natural Language Processing with Python*. 1st. O'Reilly Media, Inc., 2009. – ISBN 0596516495

[3] Chung, Junyoung ; Gulcehre, Caglar ; Cho, KyungHyun ; Bengio, Yoshua: *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling.* 2014

[4] Devlin, Jacob ; Chang, Ming-Wei ; Lee, Kenton ; Toutanova, Kristina: *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* 2018

[5] Ganesan, Kavita ; Zhai, ChengXiang ; Han, Jiawei: Opinosis: a graph-based approach to abstractive summarization of highly redundant opinions. In: *Proceedings of the 23rd International Conference on Computational Linguistics* Association for Computational Linguistics (Veranst.), 2010, S. 340–348

[6] He, Kaiming ; Zhang, Xiangyu ; Ren, Shaoqing ; Sun, Jian: *Deep Residual Learning for Image Recognition.* 2015

[7] Hendrycks, Dan ; Gimpel, Kevin: *Gaussian Error Linear Units (GELUs).* 2016

[8] Hochreiter, Sepp: The Vanishing Gradient Problem during Learning Recurrent Neural Nets and Problem Solutions. In: *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.* 6 (1998), April, Nr. 2, S. 107–116. – URL https://doi.org/10.1142/S0218488598000094. – ISSN 0218-4885

[9] Liu, Yinhan ; Ott, Myle ; Goyal, Naman ; Du, Jingfei ; Joshi, Mandar ; Chen, Danqi ; Levy, Omer ; Lewis, Mike ; Zettlemoyer, Luke ; Stoyanov, Veselin: *RoBERTa: A Robustly Optimized BERT Pretraining Approach.* 2019

[10] Mannor, Shie ; Peleg, Dori ; Rubinstein, Reuven: The Cross Entropy Method for Classification. In: *Proceedings of the 22nd International Conference on Machine Learning*. New York, NY, USA : Association for Computing Machinery, 2005 (ICML '05), S. 561–568. – URL https://doi.org/10.1145/1102351.1102422. – ISBN 1595931805

[11] Paszke, Adam ; Gross, Sam ; Chintala, Soumith ; Chanan, Gregory ; Yang, Edward ; DeVito, Zachary ; Lin, Zeming ; Desmaison, Alban ; Antiga, Luca ; Lerer, Adam: Automatic differentiation in PyTorch. (2017)

[12] Petroni, Fabio ; Rocktäschel, Tim ; Riedel, Sebastian ; Lewis, Patrick ; Bakhtin, Anton ; Wu, Yuxiang ; Miller, Alexander: Language Models as Knowledge Bases? In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Hong Kong, China : Association for Computational Linguistics, November 2019, S. 2463–2473. – URL https://www.aclweb.org/anthology/D19-1250

[13] Radford, Alec ; Wu, Jeff ; Child, Rewon ; Luan, David ; Amodei, Dario ; Sutskever, Ilya: Language Models are Unsupervised Multitask Learners. (2019)

[14] Rajpurkar, Pranav ; Jia, Robin ; Liang, Percy: *Know What You Don't Know: Unanswerable Questions for SQuAD*. 2018

[15] Sanh, Victor ; Debut, Lysandre ; Chaumond, Julien ; Wolf, Thomas: *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter*. 2019

[16] Schuster, Mike ; Nakajima, Kaisuke: JAPANESE AND KOREAN VOICE SEARCH. In: *Proc. ICASSP*, 2012, S. 5149–5152

[17] Tang, Yichuan: *Deep Learning using Linear Support Vector Machines*. 2013

[18] Vaswani, Ashish ; Shazeer, Noam ; Parmar, Niki ; Uszkoreit, Jakob ; Jones, Llion ; Gomez, Aidan N. ; Kaiser, Lukasz ; Polosukhin, Illia: *Attention Is All You Need*. 2017

[19] Wang, Alex ; Singh, Amanpreet ; Michael, Julian ; Hill, Felix ; Levy, Omer ; Bowman, Samuel R.: *GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding*. 2018

[20] Wolf, Thomas ; Debut, Lysandre ; Sanh, Victor ; Chaumond, Julien ; Delangue, Clement ; Moi, Anthony ; Cistac, Pierric ; Rault, Tim ; Louf, R'emi ; Funtowicz, Morgan ; Brew, Jamie: HuggingFace's Transformers: State-of-the-art Natural Language Processing. In: *ArXiv* abs/1910.03771 (2019)

[21] Wu, Yonghui ; Schuster, Mike ; Chen, Zhifeng ; Le, Quoc V. ; Norouzi, Mohammad ; Macherey, Wolfgang ; Krikun, Maxim ; Cao, Yuan ; Gao, Qin ; Macherey, Klaus ; Klingner, Jeff ; Shah, Apurva ; Johnson, Melvin ; Liu, Xiaobing ; Kaiser Łukasz ; Gouws, Stephan ; Kato, Yoshikiyo ; Kudo, Taku ; Kazawa, Hideto ; Stevens, Keith ; Kurian, George ; Patil, Nishant ; Wang, Wei ; Young, Cliff ; Smith, Jason ; Riesa, Jason ; Rudnick, Alex ; Vinyals, Oriol ; Corrado, Greg ; Hughes, Macduff ; Dean, Jeffrey: *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation.* 2016

[22] Yang, Zhilin ; Dai, Zihang ; Yang, Yiming ; Carbonell, Jaime ; Salakhutdinov, Ruslan ; Le, Quoc V.: *XLNet: Generalized Autoregressive Pretraining for Language Understanding.* 2019

[23] Yosinski, Jason ; Clune, Jeff ; Bengio, Yoshua ; Lipson, Hod: *How transferable are features in deep neural networks?* 2014

[24] Zeiler, Matthew D.: *ADADELTA: An Adaptive Learning Rate Method.* 2012

[25] Zhu, Yukun ; Kiros, Ryan ; Zemel, Richard ; Salakhutdinov, Ruslan ; Urtasun, Raquel ; Torralba, Antonio ; Fidler, Sanja: *Aligning Books and Movies: Towards Story-like Visual Explanations by Watching Movies and Reading Books.* 2015

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Text-Klassifikation durch BERT-basiertes Text-Splitting gesteuert durch einen Suchalgorithmus

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

_____	_____	[redacted]
Ort	Datum	Unterschrift im Original