Fachhochschule
Dortmund
University of Applied Sciences and Arts

Projektarbeit

Sentiment Analysis von Wirtschaftsnachrichten mit Deep Learning

An der Fachhochschule Dortmund

im Fachbereich Informatik

erstellte Projektarbeit

im Studiengang Informatik Vertiefungsrichtung: Praktische Informatik

zur Erlangung des Grades *Bachelor of Science in Informatik*

von

Benedikt Willecke

geboren am 15.01.1998

(Matr.-Nr.: 7105861)

Betreuer: Prof. Dr. Sebastian Bab

Bearbeitungszeit: 15.03.2020 - Heute

Eidesstattliche Erklärung

Ich versichere die von mir vorgelegte Arbeit selbständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

_____ _____

Ort, Datum Unterschrift

Zusammenfassung

In der Finanzwelt werden verschiedene Metriken genutzt, um auf eine möglichst optimale Art und Weise Investitionen zu tätigen. Hierbei spielt auch die Marktstimmung eine wichtige Rolle. Natural Language Processing ermöglicht es hierbei Textdaten automatisiert zu verarbeiten und mit Sentiment Analysis diese nach ihrer Stimmung zu klassifizieren.

In dieser Arbeit wird nach Betrachtung verschiedener Lösungsansätze das Pre-Trained Language Modell BERT durch Fine-Tuning auf Nachrichten aus der Wirtschaft abgestimmt. Es wurden verschiedene Parameter und Datensätze eingesetzt, um zu untersuchen wie das Machine Learning Modell sich bei diesen verhält und, um ein performantes sowie für den Bereich Wirtschaft allgemein anwendbares Modell zu entwickeln.

Hierbei wurde unter anderem festgestellt, dass BERT bei großen Datensätzen unempfindlich auf Parameterveränderungen reagiert, für das Fine-Tuning kleine Datensätze ausreichend sind und vielfältige Daten notwendig sind, um ein generell anwendbares Modell zu entwickeln. Das in dieser Arbeit entwickelte Modell erreicht eine Genauigkeit in der Klassifizierung in drei Sentiments von über 80% für Wirtschaftsnachrichten verschiedener Art wie Nachrichten über Ankündigungen, Quartalsberichte oder Weltgeschehen.

Weiterführende Forschung wäre jedoch von Interesse, da zu untersuchen ist, ob mit mehr Ressourcen und einem weiterentwickelten Modell, welches mehrere Sprachen unterstützt, ein genaueres und weitreichender nutzbares Modell entwickelt werden könnte.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1: Einleitung

Algorithmen beeinflussen längst viele Teile der Gesellschaft, Politik und Wirtschaft. Sie treffen Entscheidungen darüber welche Produkte dem Konsumenten vorgeschlagen werden [1], welche Werbespots welches Politikers der Wähler sieht [2] und auch in welche Unternehmen investiert wird und in welche nicht. Letzteres bezeichnet man als „algorithmischen Handel“ oder auch „hochfrequenten algorithmischen Handel“. Wobei ersteres die Bestimmung von Auftragsparametern wie Ausführungszeitpunkt, Preis und Quantität durch einen Algorithmus auf Basis von Eingangsdaten beschreibt.

Dagegen wird bei dem Hochfrequenzhandel zusätzlich die Latenz der Orderübertragungen minimiert und in hoher Frequenz Aufträge autonom ausgeführt [3] [4]. Die Nutzung dieser Technologie gewinnt mehr und mehr an Relevanz und machte bereits 2012 80% des gesamten Handelsvolumens aus [5]. Das liegt daran, dass mittlerweile jede große Investmentbank ihre eigene Software nutzt, um mittels des Hochfrequenzhandels höhere Gewinne zu erzielen [6].

Hierbei werden unter anderem Marktdaten, Firmendaten und auch Stimmungsdaten von Nachrichten genutzt [4], da Neuigkeiten bspw. aus Politik und Wirtschaft die Aktienkurse direkt beeinflussen können [7]. Deswegen werde ich in dieser Projektarbeit versuchen dem Thema der Sentiment Analysis näherzukommen und diese zur Analyse von Nachrichten implementieren.

1.1 Motivation

Das Feld der Sentiment Analysis, ein Teilbereich des NLP(Natural Language Processing), dient dazu die Haltung von Personen gegenüber gewissen Dingen zu analysieren [8][9]. Man kann bspw. die negative oder positive Stimmung auf Twitter erfassen, um Schwankungen der amerikanischen Aktienindexe Dow Jones, S&P500 und NASDAQ vorherzusagen [9]. Dies kann geschehen in dem man positive und negative Emotionen erkennt oder durch komplexere Emotionen genauer differenziert zwischen glücklich, beruhigt und sicher etc. Letzteres hat zu einer höheren Genauigkeit der Vorhersage des Aktienmarkts geführt [9].

Man muss dabei bspw. unterscheiden auf welcher Ebene man die Stimmung analysiert. Man kann den ganzen Text, jeden einzelnen Satz oder die Meinung genau zu einem gewissen Aspekt/Thema als positiv, negativ oder neutral klassifizieren [9].

1.2 Zielsetzung

In dieser Projektarbeit werde ich versuchen zu einem tiefen Verständnis der Sentiment Analysis zu gelangen, die verschiedenen Konzepte und Techniken dieser zu verstehen aber auch ihre Herausforderungen offenzulegen. Darüber hinaus wird der Kern des Projekts sein zu analysieren welche Art von Sentiment Analysis am besten zur Auswertung von Nachrichten geeignet sind und diese dann zu implementieren. In einer weiterführenden Bachelorarbeit könnte dann ein Algorithmus entwickelt werden, welcher diese Auswertungsergebnisse von Nachrichten nutzt, um, in Kombination mit Markt- und Geschäftsdaten, Aktienkursvorhersagen zu tätigen.

1.3 Methodik

Zuerst werde ich Quellen wie Fachbücher und Forschungspapiere nutzen, um einen Überblick über wesentliche Konzepte, Definitionen sowie Probleme der Sentiment Analysis zu schaffen. Hierbei soll auch auf die Sentiment Analysis mit Machine Learning eingegangen werden. Auf die Erläuterung von Themen, die nicht direkt das Thema Sentiment Analysis tangieren (wie Grundlagen des Machine Learning), wird verzichtet. Anschließend werde ich Vor- und Nachteile verschiedener Lösungsansätze ermitteln und dann einen dieser implementieren. Zum Schluss möchte ich die Ergebnisse meines Projektes präsentieren und ein Fazit sowie eine Aussicht auf zukünftige Arbeiten geben.

1.4 Kapitelübersicht

Kapitel 2 Grundlagen

Die für dieses Projekt relevanten Aspekte der Sentiment Analysis werden erklärt.

Kapitel 3 Lösungsansätze

Mögliche Lösungswege werden erkundet.

Kapitel 3 Planung

Die Lösungsansätze werden verglichen und ein Implementierungsansatz entwickelt.

Kapitel 4 Umsetzung

Die Implementierung wird vorgestellt und dokumentiert.

Kapitel 5 Experimente

Untersuchungen zur Implementierung werden erklärt und dargelegt.

Kapitel 6 Fazit

Die Ergebnisse der Projektarbeit werden zusammengefasst, es wird ein Fazit gezogen und ein Ausblick auf zukünftige Arbeiten dargelegt.

Kapitel 2: Grundlagen

Sentiment Analysis oder auch Opinion Mining genannt bezeichnet die Analyse nach Meinungen, Haltungen oder Emotionen von Personen bezogen auf Dinge wie Produkte, Dienstleistungen, Organisationen, Individuen, Ereignisse, Probleme oder Themen [9].

Sentiment bedeutet hier ein positives oder negatives Gefühl, welches durch den Text explizit ausgedrückt oder impliziert wird [9].

Die Sentiment Analysis versucht dieses Gefühl sowie auch das, auf das sich das Gefühl bezieht, zu identifizieren [9].

Forschung und Anwendung der Sentiment Analysis bezieht sich meist auf geschriebenen Text und somit ist dieses Themengebiet eine Untergruppe des NLP (Natural Language Processing). Aber auch auf audiovisueller Ebene findet die Sentiment Analysis ihre Anwendung [10].

Die Historie der Sentiment Analysis geht mindestens zurück auf das Jahr 2003 in dem der Begriff „Sentiment Analysis“ wohl das erste Mal auftauchte [15]. Forschung über Sentiment des Aktienmarkts begannen jedoch schon Anfang 2000 [16]. Zur gleichen Zeit wurden zu dem allgemeinen Thema der Textklassifizierung, sogar mit Hilfe von Machine Learning, Forschungen veröffentlicht [17]. Forschungen zum Verständnis von Text gehen auf den Beginn der 90er Jahren zurück [18]. Auch wenn die Sentiment Analysis in ihren Grundzügen kein neues Thema ist, so gewinnt es mehr und mehr an Popularität und Relevanz. Das liegt zum einen an den großen Datenmengen die mehr und mehr zur Verfügung stehen [13] und zum anderen an dem Einsatz von Deep Learning, welcher heutzutage weitverbreitet ist [19]. Hauptsächlich findet Forschung zur Sentiment Analysis durch Deep Learning über die Textklassifizierung von Beiträgen in sozialen Netzwerken statt, sei es Filmrezensionen,

Tweets oder andere Beiträge [19]. Aber auch in anderen Bereichen findet die Sentiment Analysis ihre Anwendung.

2.1 Anwendungsgebiete

Für Unternehmen, Organisationen und Regierungen ist die Meinung der Konsumenten bzw. die öffentliche Meinung sehr wichtig [9]. Anstatt mit Umfragen, oder anderen Methoden der Marktforschung, ist es nun mit Hilfe der Sentiment Analysis möglich diese Meinungen zu ermitteln, da die benötigten Informationen dazu öffentlich zugänglich sind [9]. Wie bspw. auf Twitter geteilte Meinungen über Entscheidungen der Politik oder auf Amazon veröffentlichte Produktbewertungen.

Auch im Finanzsektor findet die Sentiment Analysis ihre Anwendung. So hat die größte Bank der USA JPMorgan Chase einen Algorithmus entwickelt, welcher anhand von 250000 Analystenberichten gelernt hat, diese nach positiv, negativ und neutral zu bewerten. Diesen testeten sie an 100000 Nachrichtenartikeln, um Unterstützung bei Kaufentscheidungen zu haben und konnten somit am globalen Aktienmarkt bessere Gewinne erzielen als einige Aktienindexe [11].

Auch der größte unabhängige Vermögensverwalter BlackRock nutzt in seiner Portfoliomanagementsoftware Aladdin die Sentiment Analysis, um jedes Quartal mehr als 5000 Telefonkonferenzen zu Finanzergebnissen sowie mehr als 6000 Analystenberichte täglich zu analysieren [12].

2.2 Ebenen der Analyse

Es gibt hauptsächlich drei Ebenen, auf denen die Sentiment Analysis durchgeführt werden kann.

Document Level

Die Analyse auf Dokumentenebene bedeutete, dass man klassifizieren will, ob ein ganzes Dokument eine positive oder negative Meinung widerspiegelt. Dies kann z.B. Anwendung bei der Bewertung von Produktrezensionen finden. Hierbei wird jedoch angenommen, dass das

jeweilige Dokument lediglich eine Meinung ausdrückt, weshalb eine Analyse auf dieser Ebene nicht fein genug für Texte mit mehreren Entitäten ist [9].

Sentence Level

Hierbei wird untersucht, ob ein Satz eine positive, negative oder neutrale Meinung ausdrückt oder impliziert. Dabei gilt die gleiche Annahme wir auf dem Document Level [9].

Aspect Level

Wenn man wissen möchte worauf sich die positive oder negative Meinung bezieht muss man die Analyse auf Aspektebene durchführen. Der Satz „Auch wenn die Akkulaufzeit etwas kurz ist, mag ich das iPhone 10“ drückt eine positive Meinung in Bezug auf das iPhone aus, jedoch nicht auf seine Akkulaufzeit. Auf Satz- oder Dokumentenebene könnte man diesen Satz deswegen nicht genau positiv oder negativ bewerten. Die Analyse bezogen auf Aspekte ermittelt auf was sich die jeweilige Meinung bezieht und erhöht somit die Genauigkeit der Klassifizierung. Hierbei wird zwischen Entität und Aspekt unterschieden. In dem vorherigen Beispiel wird das iPhone 10 als Entität bezeichnet, da sich eine Meinung auf dieses bezieht. Die Akkulaufzeit hingegen gilt als Aspekt, weil sie Teil des iPhone 10 ist. Obwohl hier auch eine Meinung über die Akkulaufzeit ausgedrückt wird, kann diese bei Analysen vernachlässigt werden, falls nur die Meinung über das Gesamtprodukt, also die Entität, von Relevanz ist [9].

2.3 Ziel der aspektorientierten Analyse

Datenrepräsentation

Bei der Analyse wird versucht das Sentiment einer Meinung zu ermitteln. Diese wird wie folgt definiert: Sie besteht aus einer Entität „e“ und dem Aspekt einer Entität „a“, auf die sich jeweils eine Meinung bezieht (Target). Aus der Meinung selbst, ob positiv oder negativ, also dem Sentiment „s“. Auch ist es wichtig zu wissen wer diese Meinung inne hält, genannt Opinion Holder „h“. Sowie den Zeitpunkt der Meinungsäußerung „t“. So lässt sich folgendes

Tupel gestalten: (a, e, s, h, t). Hierbei ist zu beachten, dass Entitäten sowohl eine Hierarchie von Eigenschaften wie auch von Teilen haben. Hierdurch wird eine höhere Genauigkeit auf Kosten von höherer Komplexität erzielt. In der Umsetzung ist dies jedoch meist zu komplex und die Genauigkeit wird in den meisten Anwendungsfällen nicht benötigt, weshalb eine Beschränkung auf eine Entität mit verschiedenen Aspekten sinnvoll ist [9].

Bewertung von Sentiment

Das Sentiment der Entitäten oder auch der Aspekte wird nach positiv, negativ oder auch neutral bewertet. Diese Bewertung wird auch Sentiment Orientation (SO) genannt. Hier kann auch nach der Intensität, also der Sentiment Intensity, unterschieden werden. Diese wird in der Praxis meist durch ein Sentiment Rating, die Bewertung des Sentiments mit diskreten Zahlen, umgesetzt. Jedoch ist es meist ausreichend durch positiv (bezeichnet als +1), neutral (bezeichnet als 0) und negativ (bezeichnet als -1) zu unterscheiden [9].

Zielsetzung

Das Ziel ist es zu Dokument D alle Teile des Tupels (a, e, s, h, t) zu ermitteln, das Sentiment der Entität bzw. der Aspekte zu bewerten und schlussendlich eine Zusammenfassung zu generieren. Die Vorgehensweise ist wie folgt:

1. Alle Entitäten extrahieren.
2. Alle Aspekte extrahieren.
3. Die Meinung einer Person zuordnen also den Holder extrahieren.
4. Den Zeitpunkt der Veröffentlichung ermitteln und diesen standardisieren.
5. Sentiment aller Entitäten und Aspekte bewerten.
6. Mit diesen Daten das Tupel erstellen.

Nachdem alle Tupel zu einem Dokument erstellt wurden werden diese zusammengefasst. Es werden die insgesamte, sowie pro Aspekt/Entität, Häufigkeit der positiven, neutralen und negativen Bewertungen ermittelt. Diese können auch prozentual angegeben werden.

Bei der Extraktion von Entitäten und Aspekten ist noch zu beachten, dass diese gruppiert werden müssen. Das heißt, dass Synonyme nicht als unterschiedliche Entitäten gedeutet werden, sondern als eine. So sind „Foto“ und „Bild“ zwar unterschiedliche Worte, jedoch werden sie als eine Entität gruppiert und bewertet [9].

2.4 Emotionen und Stimmungen

Definition

Anstatt eine Meinung mit Hilfe von dem Sentiment nur als positiv, negativ und neutral zu bewerten können wir diese auch in unterschiedliche Emotionen oder Stimmungen kategorisieren. Emotionen sind hierbei kurzfristige und Stimmungen langfristige Gefühle. Hierbei wird in primäre, sekundäre und tertiäre Gefühle unterschieden, wobei diese mit steigender Genauigkeit aber auch Komplexität einhergehen [9].

Erkennung

Erkennen kann man Emotionen und Stimmungen mit sechs Mitteln, genannt Affects. Diese sind Ausdrücke von Emotionen [9].

1. Emotionale Wörter wie „lieben“ und „wütend“.
2. Sätze, die emotionales Verhalten beschreiben: „Er hat geweint als er gehört hat, dass seine Mutter gestorben ist“.
3. Adverbien wie „sehr“ und „total“.
4. Benutzung von Superlativen: „Das ist einfach das beste Auto“.
5. Verwendung von Pejorativ (z.B. „Er ist ein Faschist.“), Laudativ (z.B. „Er ist ein Heiliger.“) und Sarkasmus (z.B. „Was ein tolles Auto, es ist schon am zweiten Tag.“).
6. Verwendung von Schimpfwörtern, Anschuldigungen oder Drohungen

Datenrepräsentation

Die zu ermittelnden Daten können auch hier mit einem Tupel dargestellt werden. Auch hier benötigen wir eine Target Entität „e“ mit einem dazugehörigen Aspekt „a“. Zusätzlich benötigen wir einen Emotionstyp „m“ und eine Person „f“, die diese Emotion fühlt zu einem Zeitpunkt „t“. So erhält man das folgende Tupel: (e, a, m, f, t). Falls diese benötigt ist, kann man auch zusätzlich das Sentiment ermitteln [9].

2.5 Meinungen

Bei der Analyse von Meinungen lassen sich verschiedene Dinge weiter differenzieren.

Reguläre und vergleichende Meinungen

Die reguläre Meinung kann auch einfach als Meinung bezeichnet werden. Hierbei gibt es zwei Unterkategorien. Die direkte und indirekte Meinung. Bei ersterem wird direkt über eine Entität oder Aspekt eine Meinung ausgedrückt. Bei letzterem geschieht dies nur indirekt. Beispielsweise drückt der Satz „Nach Einnahme der Tablette habe ich mich noch schlechter gefühlt.“ indirekt auch eine negative Sichtweise auf die Tablette aus. Da dies jedoch schwer zu ermitteln ist wird sich meistens auf die Analyse von regulären Meinungen beschränkt.

Wie der Name schon vermuten lässt, handelt es sich bei der vergleichenden Meinung um einen Vergleich zwischen zwei Entitäten bzw. Aspekten [9].

Subjektive und durch Fakten implizierte Meinungen

Bei Meinungen werden außerdem zwischen subjektive und durch Fakten implizierte Meinungen unterschieden. Ersteres wird in drei Typen klassifiziert: Affect, Judgement und Appreciation. Bei Affect handelt es ich um Meinungen über Emotionen, bei Judgement über intelligente Entitäten wie bspw. Personen und bei Appreciation handelt es sich um Meinungen über nicht-intelligente Dinge wie Ästhetik [9]. Im Kontext dieser Projektarbeit sind aber lediglich die durch Fakten implizierten Meinungen relevant, da es um nicht-intelligente Dinge geht wie Wirtschaftswachstum, Umsätze von Unternehmen oder Arbeitslosenquote. Bei durch Fakten implizierte Meinungen, im Englischen „Fact-implied opinion“, handelt es sich also um eine reguläre oder vergleichende Meinung, bei der eine Meinung von einer objektiven oder faktenbasierten Aussage getätigt wird [9]. Hierbei wird weiter in zwei Unterkategorien unterteilt:

1. **Personal fact-implied opinion**, zu Deutsch persönliche, durch Fakten implizierte, Meinung. Hierbei handelt es sich um eine Meinung, welche auf persönlicher Erfahrung basiert wie z.B. „Mein Vater hat gestern ein Auto gekauft und es ist heute schon kaputt gegangen.“. Hier ist jedoch zu beachten, dass obwohl diese Art der Meinung ein positives

oder negatives Sentiment impliziert und obwohl diese auch faktenbasiert ist, ist sie im Grunde nicht anders als eine subjektive Meinung [9].

2. Anders ist da die **Nonpersonal fact-implied opinion**. Bei dieser Art der Meinung wird lediglich über einen Fakt berichtet und keine Meinung von jemandem abgegeben. Wie z.B. bei dem Satz „Der Umsatz von Google ist um 30% gestiegen.“. Hierbei ist dies nie eine persönliche Meinung, weshalb es auch keinen Opinion Holder (eine Person, die diese Meinung vertritt) gibt. Es gibt lediglich eine Quelle für diesen Fakt. Trotzdem können auch diese Art der Sätze als Meinung behandelt werden. Das liegt daran, dass durchaus die Sätze eine positive oder negative Situation widerspiegeln. Sogar der Grund dafür, dass diese Information veröffentlicht wird impliziert, dass die Aussagen positiv oder negativ sind [9].

Perspektive

In manchen Situationen kann es auch wichtig sein zu ermitteln, ob die Meinung von dem Erzähler oder von einer dritten Person ausgeht. Beispielsweise wenn der Autor die Meinung eines Analysten oder des Unternehmens selbst zu den Quartalszahlen wiedergibt ist es wichtig zu unterscheiden, ob die Meinung in der Ich-Perspektive geschrieben ist, also die Meinung des Autors ist, oder ob es die Meinung Dritter ist.

Auch kann es wichtig sein die Perspektive des Lesers zu differenzieren. Z.B. bei einem Bericht über die Steigerung des Umsatzes von Google ist dies positiv für Leser, die in Google investiert sind, jedoch nicht für Googles Konkurrenz.

2.6 Sentimentlexikon

Das Sentimentlexikon beschreibt ein Lexikon, bestehend aus Wörtern und Redewendungen mit ihrem jeweilig zugeordneten positivem oder negativem Sentiment, welches sie ausdrücken. Diese Wörter bzw. Redewendungen werden auch als Sentiment Words oder auch Opinion Words genannt. Beispielsweise drückt das Wort „gut“ eine positive und das Wort „schlecht“ eine negative Haltung aus [9]. Hierbei treten jedoch folgende Probleme auf:

1. Das Sentiment der Wörter kann vom Kontext abhängen.

Der Satz „Das Messer ist sehr scharf“ zeigt die positive Qualität des Messers auf. „Die Kanten vom Schreibtisch sind sehr scharf“ hingegen drückt eine nicht gewollte und möglicherweise gefährliche Eigenschaft des Tisches aus [9].

2. Ein Satz mit Opinion Words kann auch neutral sein.

Fragen wie „Können Sie mir eine gute Sony Kamera empfehlen?“ haben weder ein positives oder negatives Sentiment obwohl das positive Sentiment Wort „gut“ vorkommt [9].

3. Sarkasmus kann zu falschen Klassifizierungen führen.

Sarkastische Sätze wie „Das Auto war so gut, dass es schon nach zwei Tagen nicht mehr funktioniert hat!“ könnte als positiv von einem Algorithmus klassifiziert werden, da Sarkasmus noch eine große Herausforderung darstellt [9].

4. Sätze ohne Opinion Words können auch eine Haltung implizieren.

Obwohl in dem Satz „Die Waschmaschine benutzt sehr viel Wasser“ keine positiven oder negativen Sentiment Words enthalten sind, drückt es jedoch eine negative Eigenschaft des Produktes aus [9].

Kapitel 3: Lösungsansätze

In den folgenden Abschnitten werde ich die verschiedenen Algorithmen der Sentiment Analysis als Lösungsansätze vorstellen und diskutieren, welchen ich implementieren möchte. Zuerst werde ich darauf eingehen wie man auf den verschiedenen Ebenen der Analyse die Sentiments klassifizieren kann und welche Rolle Machine Learning dabei spielt. Außerdem werde ich die verschiedenen Techniken vorstellen, die bei der Sentiment Analysis genutzt werden können.

3.1 Document Sentiment Classification

Die Document Sentiment Classification ist die Klassifizierung des Sentiments einer Meinung auf Dokumentenebene nach positiv, negativ oder neutral. Hierbei wird das Dokument allgemein betrachtet und es wird nicht auf Entitäten oder Aspekte untersucht. Es gilt als die einfachste Art der Sentiment Analysis. Um aussagekräftige Ergebnisse zu erlangen gibt es jedoch einige Voraussetzungen: Es wird angenommen, dass das Dokument zu einer einzigen Entität genau eine Meinung, eines einzelnen Opinion Holders, enthält. Dies hat zur Folge, dass diese Art der Klassifizierung nur in speziellen Anwendungsfällen genutzt werden kann [9].

Anwendungsfall

Meist wird die Document Sentiment Classification bei Produktrezensionen angewendet, da man hier von einem einzigen Opinion Holder, der Stellung zu einem einzigen Produkt nimmt, ausgehen kann. Gerade diese benötigen aber meist keine Sentiment Analysis, da sie schon eine Sternebewertung o.ä. haben und somit Informationen darüber, ob der Opinion Holder positiv, negativ oder neutral über das Produkt denkt. Auf der anderen Seite ist es schwierig Dinge wie Forumsdiskussionen, Blogs oder Nachrichtenartikel lediglich auf Dokumentenebene zu analysieren, da es sich hierbei meist um Texte mit mehreren Entitäten oder auch Opinion Holder handelt. Besonders in solchen Fällen wird die Sentiment Analysis aber gebraucht [9].

3.2 Sentence Sentiment Classification

Die Sentence Sentiment Classification ermittelt das Sentiment jedes einzelnen Satzes und bietet so mehr Genauigkeit als die Document Sentiment Classification. Es wird jedoch weiterhin angenommen, dass jeder Satz genau ein Sentiment ausdrückt, ob positiv, negativ oder neutral [9].

Anwendungsfall

Auch wenn die Sentence Sentiment Classification genauere Ergebnisse liefern kann als die Document Sentiment Classification, so sind diese Ergebnisse nicht immer genau genug. Der Grund dafür ist der, dass bei der Sentence Sentiment Classification die Annahme besteht, dass jeder Satz nur ein Sentiment ausdrückt. Dadurch kann diese Klassifizierungsmethode nur bei simplen Sätzen eingesetzt werden. Nachrichten bspw. enthalten aber oft komplexe Sachzusammenhänge mit verschiedenen Themen und Sentiments. Ein Grund dafür ist auch, dass diese Klassifizierungsart vergleichende Sätze nicht richtig bestimmen kann, da den verschiedenen Entitäten bzw. Aspekte nicht unterschiedliche Sentiments in einem Satz zugeordnet werden können [9].

3.3 Aspect Sentiment Classification

Die Aspect Sentiment Classification versucht das Sentiment zu jeder Entität bzw. jedem Aspekt zu ermitteln. Das hat den Vorteil, dass komplexe Sätze korrekt klassifiziert werden können. So kann bspw. zwischen Apple, Microsoft und dem Technologiesektor allgemein, die alle im selben Kontext auftauchen können, differenziert werden. Um einen Sentiment Analysis Algorithmus zu implementieren existieren zwei grundsätzliche Herangehensweisen: Supervised oder Unsupervised [9]. Letzteres wird auch in diesem Kontext als Lexicon-Based-Ansatz oder Dictionary-Based-Ansatz bezeichnet. Zudem findet auch Deep Learning mehr und mehr Verwendung in der Sentiment Analysis.

Unsupervised Ansatz

Bei dieser Lösungsmöglichkeit wird ein Sentiment Lexikon benutzt, welches eine Liste an Wörtern und Redewendungen mit ihrem dazugehörigen Sentiment enthält. Im Folgenden

erläutere ich den Ablauf dieser Methode. Hierbei gilt jedoch die Annahme, dass die Aspekte und Entitäten bekannt sind. Die Aspekt Extraktion wird in einem späteren Abschnitt vorgestellt [9]. Grundsätzlich funktioniert dieser Lösungsansatz folgendermaßen: Jedem Sentiment Ausdruck wird entsprechend dem genutzten Sentiment Lexikon ein positiver oder negativer Wert zugewiesen. Danach werden diese lediglich summiert und man erhält einen finalen Sentiment Wert für das Dokument, den Satz oder Satzteil, den man untersucht. Hierbei gibt es verschiedene Variationen, die sich darin unterscheiden welcher Wert zugewiesen wird, wie Negationen behandelt werden und ob zusätzliche Informationen benutzt werden. Grundsätzlich ist dieses System regelbasiert. So drehen Negationswörter wie „nicht“ den Sentiment Wert um. Somit sind Negationswörter eine Untergruppe der Sentiment Shifter. Diese sind Wörter, die den vorher zugeordneten Sentiment Wert verändern. So gibt es auch „intensifier“ und „deminisher“, die jeweils den Sentiment Wert verstärken oder verringern. So liegt der grundsätzliche Sentiment Wert, der zugewiesen wird, bei 2. Wird zusätzlich ein intensifier oder deminisher benutzt so wird der Wert auf 3 erhöht bzw. auf 1 verringert. Negationswörter verändern den Wert grundsätzlich um -2. In Verbindung mit einem intensifier sogar um -3 und in Verwendung zusammen mit einem deminisher nur um -1. Eine andere Variante ist folgende: Die zugewiesenen Werte reichen von -5 bis +5 (0 ist ausgeschlossen). Jeder deminisher und intensifier ist mit einem Prozentwertbelegt. So könnten „slightly“ -50, „pretty“ -10, „really“ +15, „very“ +25 und „(the) most“ +100 zugewiesen werden. Angenommen dem Wort “excellent” sei der Wert 5 zugewiesen so würde der Sentiment Wert für „most excellent“ folgendermaßen berechnet: 5 * (100% + 100%) = 10. Wenn „good“ den Wert 3 hat so hat „really very good“ den Wert (3 * [100%+25%]) * (100% + 15%) = 4,3. Diese Rechnung geschieht also rekursiv. Da die einfache Umkehrung der SO zu unpassenden Werten führen kann, kann folgendes System verwendet werden: Die SO wird um einen bestimmten in die gegenüberliegende SO verschoben z.B. um 4. So würde „not excellent“ der Wert 5 – 4 = 1 zugewiesen. Die Idee dahinter ist, dass es schwierig ist besonders positive Wörter zu negieren, weshalb man die Negierung eher wie einen „diminisher“ behandeln sollte. Es sollte auch erwähnt werden, dass lexikonbasierte Sentiment Klassifikatoren die Tendenz haben das Sentiment eher als positiv zu bewerten, da negative Ausdrücke relativ selten vorkommen. Um die Neutralität zu bewahren kann der Sentiment Wert von negativen Ausdrücken mit einem Gewicht von z.B. +50% verstärkt werden. Außerdem gibt es sogenannte „Irrealis Markers“, welche Hinweise darauf geben, dass dieser Satz nicht zuverlässig klassifiziert werden kann, obwohl der Satz ein Sentiment ausdrückt. Darunter gehören Fragen, Zitate, Verben wie „erwarten“ oder „anzweifeln“ und

Bedingungswörter wie „falls“. Das Ignorieren solcher Sätze wird als „Irrealis Blocking“ bezeichnet [9].

Bei der Anwendung des Lexicon Based Ansatzes auf die Aspect Sentiment Classification müssen wir dieses Verfahren pro Aspekt/Entität anwenden. So kommt man auf folgende Beschreibung des Algorithmus:
1. Alle Sentiment Ausdrücke, also Wörter und Redewendungen, als positive oder negative Ausdrücke markieren [9].

2. Anwendung der Sentiment Shifter. ZB. Negierungswörter [9].

3. Behandlung von Wörtern wie „aber“, die auf einen Widerspruch deuten. In dem Satz „Das iPhone ist gut, aber die Akkulaufzeit ist kurz.“ erhält der erste Teil des Satzes ein positives Sentiment zugeteilt. Durch das „aber“ lässt sich implizieren, dass das „kurz“ hier ein negatives Sentiment zugeteilt werden muss. Hierbei ist zu beachten, dass das Wort „aber“ nicht immer auf einen Widerspruch hindeutet wie in dem Fall „aber auch“ [9].

4. Im letzten Schritt wird die Sentiment Aggregation Function ausgeführt. Diese sieht wie folgt aus:

$$score(a_i, s) = \sum_{se_j \in s} \frac{se_j.ss}{dist(se_j, a_i)}$$

Der Sentiment Score jedes Aspektes in einem gegebenen Satz wird durch diese Aggregation berechnet. Wobei se_j ein Sentiment Ausdruck im Satz ist, $dist(se_j, a_i)$ die Distanz des Ausdrucks zu dem Aspekt a_i ist und $se_j.ss$ den Sentiment Score des jeweiligen Sentiment Ausdrucks darstellt. Es wird durch die Distanz zwischen dem Ausdruck und dem Aspekt geteilt, um Sentiment Ausdrücken, welche weit entfernt von dem Aspekt sind, weniger Gewicht zu geben. Das Ergebnis, ob positiv oder negativ, wird als Sentiment für den Aspekt a_i in dem Satz s zugewiesen. Falls es weder positiv noch negativ ist, so gilt es als neutral [9].

Um die Sentiment Ausdrücke ihren jeweiligen Aspekten zuzuweisen ist es auch möglich, anstatt der Distanz zwischen beiden, den Scope des Ausdruckes zu bestimmen. Hierbei kann man sich die Beziehung zwischen beiden zu Nutze machen. Diese Beziehungen lassen sich in drei Gruppen unterteilen [9]:

1. Syntaktische Abhängigkeit. Hierbei geht man davon aus, dass Adjektive und Nomen sowie Verben und Adverbien sich aufeinander beziehen. Bei dem Satz „Das interessante

Buch.“ bezieht sich das Adjektiv „interessant“ auf das Nomen bzw. den Aspekt „Buch“. Das gleiche gilt für Verb-Adverb Beziehungen [9].

2. Das Sentiment Wort selbst. Oft deuten Sentiment Wörter direkt auf das Sentiment eines Aspektes hin. So folgt aus dem Wort „teuer“ ein negatives Sentiment für den Aspekt Preis. In dem Satz „BMW ist teuer“ können wir dann durch die syntaktische Abhängigkeit das negative Sentiment des Aspektes „Preis“ der Entität „BMW“ zuweisen [9].

3. Semantische Beziehung. Hierbei wird das Verständnis des Inhaltes genutzt, um ein Sentiment zu seinem jeweiligen Target zuzuweisen. In dem Satz „Peter bewundert die deutsche Ingenieurskunst“ wird durch die inhaltliche Analyse der „deutschen Ingenieurskunst“ ein positives Sentiment zugeteilt. Hierbei ist jedoch zu erwähnen, dass es äußerst komplex ist zu so einem semantischen Verständnis zu gelangen [9].

Eine andere Methode wurde 2002 vorgestellt. Diese baut auf sogenannten Part-of-Speech (POS) Tags auf. Kurzgesagt wird jeder Wortart ein Tag zugewiesen. So ist ein Adjektiv ein JJ, ein Nomen im Singular ein NN und im Plural NNS. Um nun zu einem Sentiment Wert zu gelangen werden drei Schritte vollzogen.

1. Zwei aufeinanderfolgende Wörter werden extrahiert, wenn sie einem zuvor festgelegten Mustern entsprechen. Z.B. Das erste Wort ist ein JJ und das zweite ein NN oder NNS.

2. Die Sentiment Orientation wird mit Hilfe der Pointwise Mutual Information (PMI) geschätzt:

$$PMI(term_1, term_2) = log_2(\frac{\mathrm{P}(term_1 \wedge term_2)}{\mathrm{P}(term_1)\mathrm{P}(term_2)})$$

Der PMI Wert zeigt an wie statistisch abhängig die beiden Ausdrücke sind. Hierbei ist $\mathrm{P}(term_1 \wedge term_2)$ die tatsächliche Wahrscheinlichkeit, dass beide Begriffe zusammen vorkommen. Und $\mathrm{P}(term_1)\mathrm{P}(term_2)$ zeigt die Wahrscheinlichkeit an, dass beide gemeinsam vorkommen, angenommen sie sind statistisch unabhängig. Der SO Wert wird ermittelt, indem der PMI Wert eines Ausdrucks und jeweils einem positiven und negativen Wort verglichen wird. Hier ein Beispiel:

$$SO(phrase) = PMI(phrase, \text{"excellent"}) - PMI(phrase, "poor")$$

Die Wahrscheinlichkeiten werden durch Hits ermittelt, also die Häufigkeit wie oft ein Wort in der Nähe von „excellent“ bzw. „poor“ vorkommt. Dadurch kann die Gleichung folgendermaßen umgeschrieben werden:

$$SO(phrase) = log_2(\frac{hits(phrase\ NEAR\ \text{excellent})hits(\text{"}poor\text{"})}{hits(phrase\ NEAR\ \text{poor})hits(\text{"}excellent\text{"})})$$

3. Im letzten Schritt werden lediglich die SO Werte summiert für das jeweilige Dokument, das untersucht wird, sei es ein ganzer Text, ein Satz oder nur ein Satzteil.

Supervised Ansatz

Da es sich hier um Textklassifizierung handelt kann jeder gängige Machine-Learning Algorithmus wie Support Vector Machines (SVM) und Naive Bayes-Klassifikator angewandt werden. Das Entscheidende ist jedoch, welche Features dabei benutzt werden. Hierbei ist zu beachten, dass diese Features abhängig von der Entität bzw. dem Aspekt sind. D.h. unterschiedliche Entitäten und Aspekte haben andere Features, mit denen sie untersucht werden müssen. Hierbei gibt es zwei Lösungsansätze [9]:

1. Für jede Entität bzw. jeden Aspekt generiert man eine Liste von Features, die von diesen abhängig sind [9].

2. Andersrum kann man für jedes Feature den jeweiligen „feature scope ermitteln“. D.h., dass ermittelt wird auf welche Entitäten bzw. Aspekte sich das Feature bezieht [9].

In der Forschung wurden verschiedene Features vorgestellten. Ein paar Beispiele sind:

Bag of Words(BoW) – Das neuronale Netz braucht einen Weg den Text als Input, also als Features, zu bekommen. Hierzu nehmen wir an wir haben eine gewisse Anzahl an Sätzen. Man nimmt nun jedes Wort als Token. Daraufhin wird für jeden Satz ein Vektor erstellt, in dem die Häufigkeit jedes Tokens in dem jeweiligen Satz gezählt wird. So erhält man den Feature Vektor für den jeweiligen Satz [20].

Term Frequency-Inverse Document Frequency (TF-IDF) – Ähnlich wie in BoW wird die Häufigkeit eines Wortes betrachtet, hier jedoch in Relation zu ihrer Wichtigkeit gestellt. Dies geschieht wie folgt [20]:

$$TF(t) = \frac{Häufigkeit\ von\ Term\ t\ im\ Dokument}{Anzahl\ aller\ Terms\ im\ Dokument}$$

$$IDF(t) = log_e(\frac{Anzahl\ aller\ Dokumente}{Anzahl\ der\ Dokumente\ mit\ Term\ t})$$

$$TF - IDF_{score} = TF * IDF$$

Es wurde erwiesen, dass die Nutzung dieser Art von Feature sehr effektiv für die Sentiment Classification ist.

Part of Speech (POS) – Hierbei wird jedes Wort einer Wortkategorie zugeordnet. Dies kann genutzt werden, um genauer zu bestimmen, wie wichtig gewisse Wortarten, wie z.B. Adjektive, sind. So erhält ein Nomen das Tag NN, ein Nomen im Plural NNS und ein Adjektiv das Tag JJ [9]. Das Tagging der Worte geschieht üblicherweise mit einer POS-Tagging Software. Eine gängige ist die der Stanford University [21].

Sentiment Wörter und Redewendungen – Wörter und Redewendungen, die ein Sentiment ausdrücken bieten sich natürlicherweise als Features an, weil sie direkt Aussagen über das Sentiment geben [9].

Sentiment Shifters – Die schon beschriebenen Sentiment Shifters haben eine wichtige Auswirkung auf das Sentiment, weshalb sie auch als Features benutzt werden können [9].

Syntaktische Abhängigkeiten – Hierbei wird mit Hilfe von Syntactic Parsing, oder auch Dependency Parsing genannt, einem Satz eine syntaktische Struktur zugeteilt. Diese wird meistens in einen Syntaxbaum übergeben. So könnte aus dem Satz „This is a great car.“ folgendes gebildet werden: amod_car_great oder verallgemeinert amod_NN_great. Hierbei steht amod für einen „adjectival modifier“, also ein Adjektiv, welches etwas modifiziert. NN steht, wie aus dem POS Tagging bekannt, für ein Nomen.

Word Embeddings – Zu Deutsch Worteinbettungen, stellen Wörter jeweils als einen Punkt im Raum, also als ein Vektor, dar, in dem ähnliche Wörter näher zueinander stehen. Die bekannteste Software, die solche Word Embeddings berechnet, ist wohl Word2Vec von Google. Hierbei wurde ein Machine Learning Modell auf 100 Milliarden Wörtern trainiert. Das Modell beinhaltet 300-dimensionale Vektoren für 3 Millionen Wörter und Redewendungen. Das Resultat ist, dass diese Vektoren nicht nur syntaktische Ähnlichkeit zwischen Wörtern sondern auch semantische Ähnlichkeit widerspiegeln. Da es sich hier um Vektoren handelt lässt sich auch die Vektor-Arithmetik Anwenden. Z.B. vector(„king“) – vector(„man“) + vector(„woman“) = vector(„queen“) [27][28].

Zur schlussendlichen Klassifizierung können klassische Machine Learning Methoden wie Logistic Regression oder Support Vector Machines genutzt werden [22].

3.4 Deep Learning und Sentiment Analysis

Es wurde aber auch weiterhin versucht die Klassifizierungsgenauigkeit zu erhöhen durch die Anwendung von Deep Learning bspw. in Form von einem RNN (Recurrent Neural Network): Hauptsächlich geschieht dies über ein bidirektionales LSTM (Long Short-Term Memory) [22][23][24]. Die folgenden Abschnitte sollen eine Erklärung und eine Übersicht für relevante neuronale Netzwerkmodelle bieten.

Recurrent Neural Network (RNN) – Zu Deutsch rekurrentes neuronales Netz wurde erstmals 1990 vorgestellt, aber erst 2012 im NLP Bereich eingesetzt [30]. Anders als Feed-forward Neural Networks, erhält das neuronale Netz als Input eine Sequenz (hier Wörter oder Sätze). Jedes Wort wird nach und nach in das RNN übergeben und es entsteht jedes Mal ein Zwischenergebnis. Dieses Zwischenergebnis wird als Zustand des RRNs bezeichnet. Der darauffolgende Zustand berechnet sich durch die Gewichtung des nächsten Zustands und durch den aktuellen Zustand. Bildlich kann das RNN folgendermaßen dargestellt werden [28]:

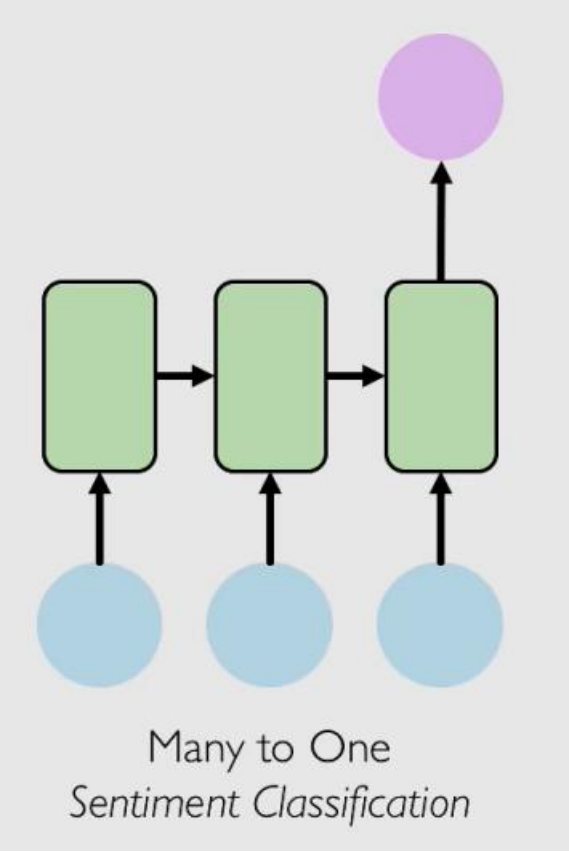


Abbildung 1: Many-to-One RNN [28]

Die verschiedenen Teile nach jedem Schritt werden rekurrente Zellen genannt und besitzen, wie oben erwähnt, jeweils einen Zustand. Dieser Zustand, auch hidden state genannt, lässt sich folgendermaßen mathematisch ausdrücken.

$$h_t = f_W(h_{t-1}, x_t)$$

h_t ist hierbei der Zellenzustand, f_W ist eine von W, der Gewichtungsmatrix, parametrisierte Aktivierungsfunktion, h_{t-1} ist der alte Zustand und x_t der aktuelle Inputvektor. Abbildung 1 zeigt ein Many-to-One RNN aber tatsächlich wird nach jedem Schritt ein Output generiert, bei Many-to-One RNNs ist nur der finale Output entscheidend für die Klassifizierung. Die Output Vektoren können auf folgende Weise berechnet werden:

$$\hat{y}_t = W_{hy}^T h_t$$

Hierbei ist W_{hy}^T die Gewichtungsmatrix vom hidden state h_t nach dem Outputvektor $\hat{y}_t$. Zuletzt können wir auch nach jedem Schritt den Verlust bzw. Loss L_t berechnen, wobei der insgesamte Loss die Summe aller L_t ist. Daraus ergibt sich folgende Struktur:

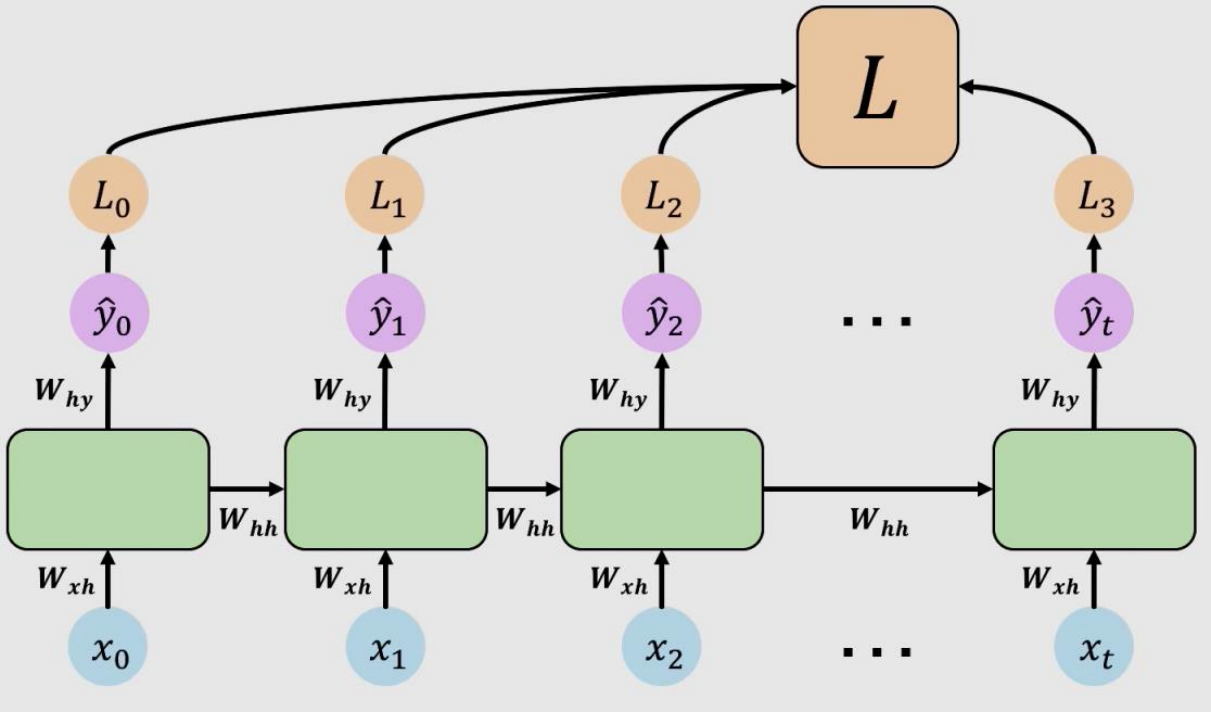


Abbildung 2: RNN mit Loss [28]

Probleme im Deep Learning:

Exploding Gradient – Bei der Optimierung eines Modells wird Backpropagation eingesetzt. Hierbei wird der jeweils Gradient der Faktoren berechnet. Bei sehr tiefen neuronalen Netzwerken kann es passieren, dass viele Gradiente, die größer als 1 sind, miteinander multipliziert werden und so sehr groß werden, sprich explodieren [28]. Dies führt zu einem

instabilen Lernprozess [34]. Gelöst werden kann dieses Problem zum Beispiel mit Hilfe von „gradient clipping“, wobei die Gradienten in einem vorher bestimmten Wertebereich gehalten werden [35].

Vanishing Gradient – Hierbei werden viele Gradiente, die kleiner als 1 sind, miteinander multipliziert, wodurch der Gradient gegen 0 geht und somit verschwindet. Dies kann ein Problem darstellen, falls Informationen gebraucht werden, die weiter hinten im Netzwerk liegen [28].

Long Short Term Memory (LSTM):

Um die Probleme des Vanishing / Exploding Gradients von RNNs zu lösen wurde eine neue Struktur, das Long Short Term Memory kurz LSTM, entwickelt und 1997 von zwei Deutschen erstmals vorgestellt [29]. LSTMs sind ähnlich wie RNNs, besitzen jedoch, anders als diese, sogenannte Gated Cells [30]. Ein Standard RNN hat lediglich eine Aktivierungsfunktion im Inneren jeder Zelle. Mit einer komplexeren Struktur innerhalb der Zelle versucht das LSTM vier Ziele zu erreichen: Irrelevante Informationen des letzten Zustandes zu vergessen, relevante neue Informationen zu speichern, selektiv den Zellenstatus zu aktualisieren und kontrolliert Informationen in die nächste Zelle übergeben [28].
1. Vergessen: Im ersten Schritt wird versucht irrelevante Informationen herauszufiltern. Hierfür wird der interne Zustand (auch history h genannt), der vorherigen Zelle, und der aktuelle Input in eine Sigmoidfunktion gegeben. Der Output ist eine Zahl zwischen 0 und 1, wobei 1 „vollständig behalten“ und 0 „vollständig vergessen“ bedeutet. Die Funktion hat zusätzlich noch eine Gewichtung und ein Bias. Dieser Schritt wird als „forget gate layer“ bezeichnet. Die Gleichung sieht wie folgt aus [28] [31]:

$$f_t = \sigma(W_t[h_{t-1}, x_t] + b_f)$$

Außerdem lässt sich dies folgendermaßen graphisch darstellen:

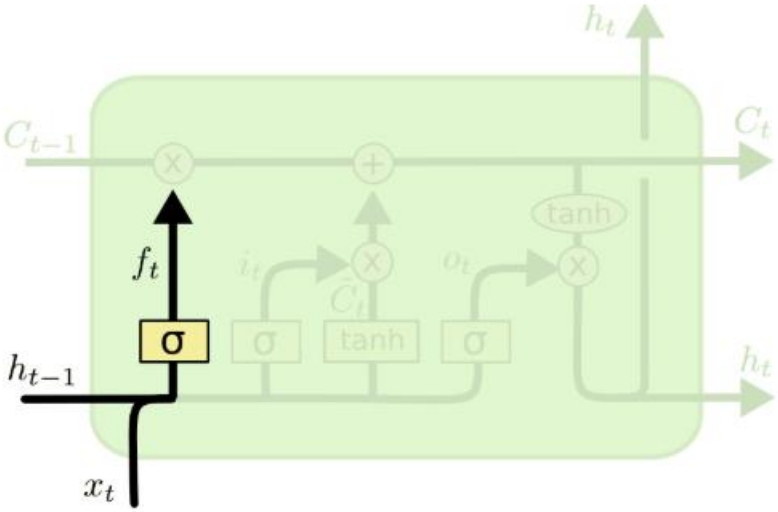


Abbildung 3: LSTM Schritt 1 – Vergessen [31]

2. Speichern: Bei diesem Schritt soll gefiltert werden, welche Informationen gespeichert werden sollen, also welche Informationen in die Aktualisierung des Zustandes beinhaltet sind. Hierbei werden zwei Ebenen durchlaufen: Zum einen wird durch eine Sigmoidfunktion entschieden, welche Werte aktualisiert werden sollen. Dieser Teil wird als „input gate layer“ bezeichnet. Als nächstes wird mit Hilfe einer tanh-Funktion ein Vektor, mit potenziellen Werten, die dem Zustand hinzugefügt werden könnten. Die beiden Gleichungen hierfür und die Grafik können wie folgt dargestellt werden [28] [31]:

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$$

$$\tilde{c}_t = \tanh(W_C[h_{t-1}, x_t] + b_C)$$

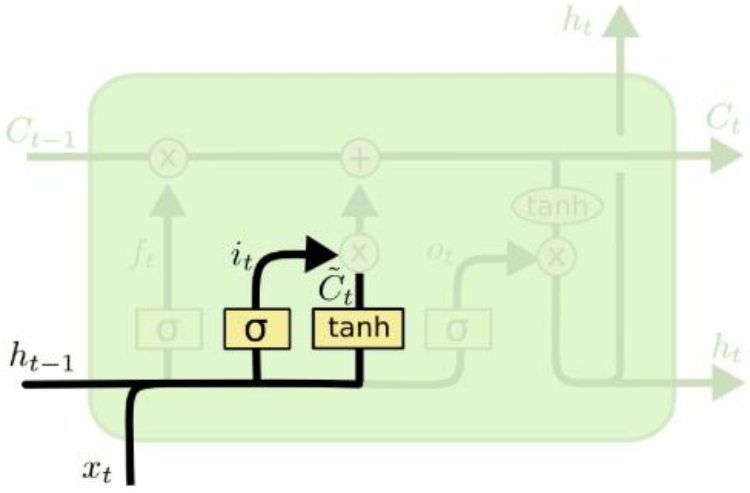


Abbildung 4: LSTM Schritt 2 – Speichern [31]

3. Aktualisieren: Nun muss der alte Zustand C_{t-1} in den aktuellen Zustand C_t überführt werden. Hierzu werden die Informationen aus den vorherigen beiden Schritten genutzt. Die beiden Zwischenergebnisse aus Schritt zwei werden miteinander multipliziert, genauso wie das Ergebnis aus Schritt 1 mit dem Zustand C_{t-1}. Die Addition dieser beiden Multiplikationen ergibt dann den neuen Zustand. Als Gleichung und Grafik sieht dies wie folgt aus [28] [31]:

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

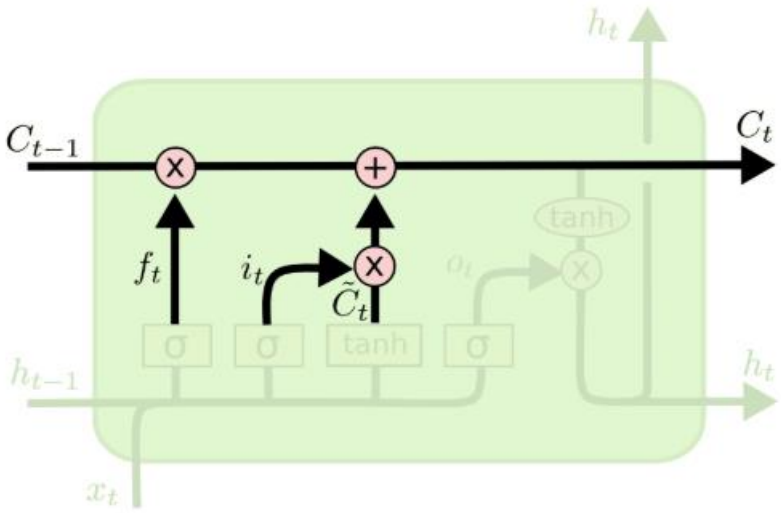


Abbildung 5: LSTM Schritt 3 – Aktualisieren [31]

4. Output: Im letzten Schritt wird entschieden, was vom Zustand als Output gewählt wird. Die Informationen des Zustandes werden also nochmals gefiltert. Hierzu wird die Sigmoidfunktion auf den vorherigen Zustand sowie den aktuellen Input angewendet und mit dem Ergebnis der tanh-Funktion des aktuellen Zustandes multipliziert. Es folgt die Gleichung und Grafik hierzu:

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

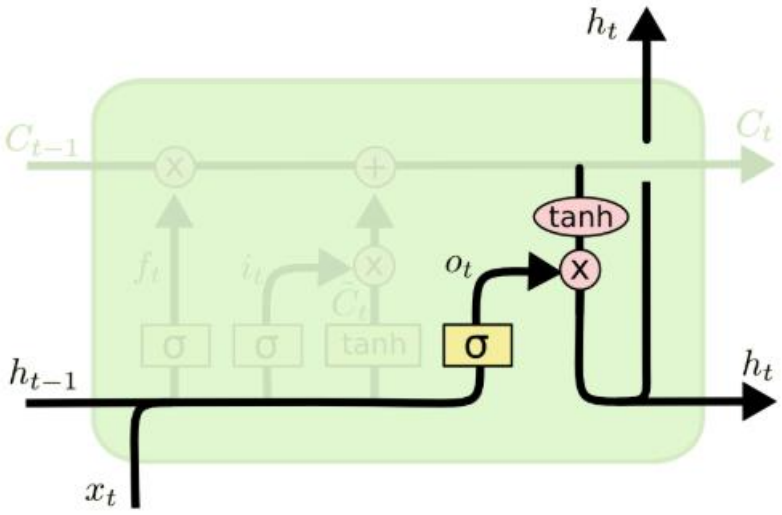


Abbildung 6: LSTM Schritt 4 – Output [31]

Varianten von LSTM – Es gibt sehr viele Varianten von LSTMs. Beispielsweise kann man in die Sigmoidfunktionen zusätzlich den aktuellen Zellenzustand geben. In einer Forschungsarbeit wurden mehr als 10.000 RNN Architekturen untersucht. Hierbei wurde festgestellt, dass einige LSTMs für bestimmte Aufgaben besser geeignet sind [32]. Wobei andere Forschungen ergaben, dass die beliebtesten LSTM Varianten mehr oder weniger gleich sind [33].

Probleme von LSTMs – LSTMs beheben einige Probleme, die RNNs haben. Namentlich Vanishing und Exploding Gradient. Wie bereits erläutert treten diese Probleme durch die sequenzielle Struktur auf. Hierbei verschwindet oder explodiert der Wert des Gradienten, da entweder kleine oder große Werte sehr oft miteinander multipliziert werden. LSTMs verringern diese Probleme durch ihre Gated Cells, beheben sie jedoch nicht vollständig, da die grundsätzliche sequentielle Struktur bestehen bleibt und so die gleichen Probleme bei sehr langen Sequenzen bzw. sehr tiefen Netzen auftreten können. Zudem sind LSTMs, und auch RNNs, im Training und der Anwendung sehr speicheraufwändig, da man pro Zelle und pro Zeitschritt vier lineare Layer braucht [38].

Pretrained Language Models

Alle neuen state-of-the-art Modelle wie ELMo, BERT oder XLNet sind sogenannte Pre-trained Language Models. Sie wurden an großen Textkorpora, wie Wikipedia, trainiert und erhalten so ein Grundverständnis von Sprache, welches durch Finetuning dann für einen

bestimmten Anwendungsbereich abgestimmt werden kann [42] [43]. BERT ist ein von Google AI Language Team entwickeltes und 2018 erschienenes Modell. Es stellt einen Meilenstein in der NLP Geschichte dar und ist der Ursprung vieler anderer ähnlicher Modelle [43] [44]. Denn durch den Einsatz von Transformern statt LSTMs konnte die Performance gegenüber früheren Modellen deutlich verbessert werden [36] [37].

Transformer – Attention is all you need

Um die Probleme der sequenziellen Architekturen von RNNs und LSTMs zu lösen, wurden 2017 Transformer mit dem Forschungspapier „Attention Is All You Need“ von Ashish Vaswani et al. vorgestellt. Durch diese Architektur ist es möglich sequenzielle Daten parallelisiert, mit Hilfe von Matrizen, zu verarbeiten, wodurch GPUs besser ausgenutzt werden können und somit sich die Trainings- und Laufzeit verbessert. Außerdem führen die Transformer einen sogenannten Attention-Mechanismus ein. Dieser legt für jedes Wort fest wie wichtig andere Worte im Kontext des Satzes für dieses Wort sind. Es wurde bewiesen, dass ganz ohne RNN-Strukturen und nur mit Attention-Mechanismen die Performance erhöht werden kann [39] [40]. Der vorgestellte Transformer sieht folgendermaßen aus:

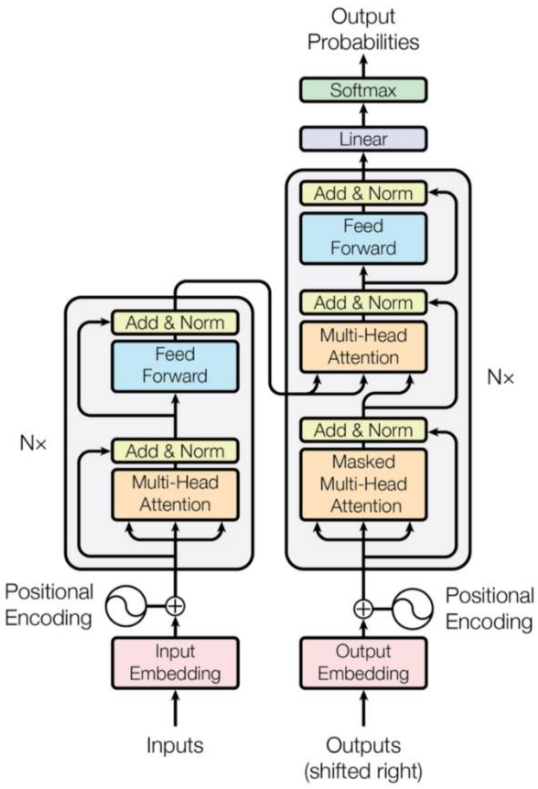


Abbildung 7: Transformer-Architektur [40]

Der linke Teil stellt hier den Encoder und der rechte den Decoder dar, weshalb man auch von einer Encoder-Decoder Struktur spricht. Die mit „Nx“ angemerkten Blöcke können beliebig oft aufeinandergestapelt werden. Wie man sehen kann besteht die Architektur hauptsächlich aus Multi-Head Attention Layern. Ein wichtiger Teil ist außerdem das Positional Encoding. Da man ohne sequenzielle RNN Struktur nicht automatisch die Position des Wortes kennt, wird die Position als Vektor mit dem Word Embedding Vektor zusammengeführt.

Encoder

In der folgenden Grafik ist der Aufbau des Multi-Head Attention Layer dargestellt [39] [40] [41]:

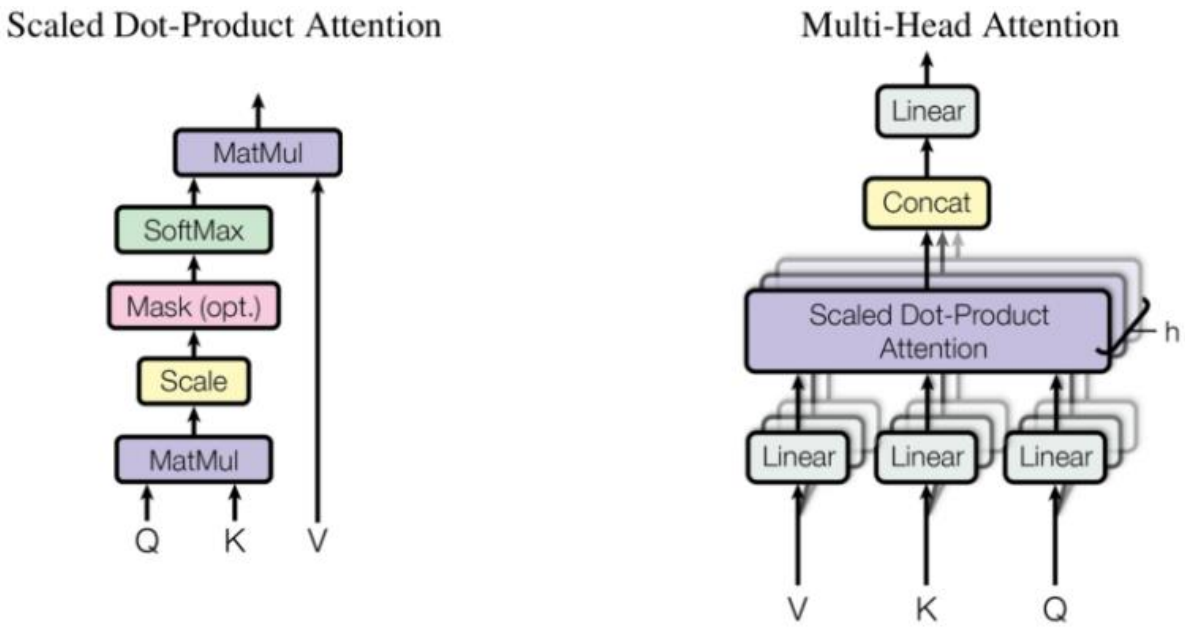


Abbildung 8: Multi-Head Attention [39]

Hierbei sind Q (Query), K (Keys) und V (Values) Abstraktionen des Inputs in Form von Matrizen, die durch eine Multiplikation der Word Embedding Matrix X und einer Gewichtungsmatrix, deren Werte im Training gelernt werden, berechnet werden:

$$X * W^Q = Q$$

$$X * W^K = K$$

$$X * W^V = V$$

Jede Zeile der Matrizen steht jeweils für den Vektor eines Wortes. Q ist die Vektorrepräsentation eines Wortes und K sowie V die Vektorrepräsentationen aller Worte. Die Dimension ist d_k. Die Attention wird dann durch folgende Gleichung berechnet:

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

Hierbei stellen höhere Werte eine größere Wichtigkeit eines Wortes in Bezug auf das Query-Wort dar. Die Softmax Funktion ist lediglich da, um die Werte in Wahrscheinlichkeiten umzuwandeln. Nach der Berechnung der Attention werden die Ergebnisse konkateniert und mit einer weiteren Gewichtungsmatrix W^O multipliziert, um das Endergebnis des Layers zu produzieren. Im Add & Norm Layer wird zuerst das Ergebnis aus dem Multi-Head Attention Layer mit dem Input von diesem, welcher durch eine Residual Connetion verbunden ist, addiert. Dies hilft dem Model beim Training, da die Gradienten so direkt durch das Netz fließen können. Die Normalisierung verhindert, dass die Spannweite der Wert sich zu sehr verändert, was zu einem schnelleren Training und einer besseren Generalisierung führt. Die Feed Forward Layer sind hingegen lediglich da, um die Matrizen in die vom nächsten Layer akzeptierte Form umzuwandeln [62].

Decoder

Der Decoder funktioniert sehr ähnlich wie der Encoder. Die Schritte im Decoder werden solange ausgeführt, bis der Decoder seinen Output vollständig ausgegeben hat. Der Anfang und das Ende werden durch spezielle Symbole angezeigt. Die jeweiligen Zwischenergebnisse werden wie beim Encoder nach oben weitergegeben. Auch die Word Embeddings mit den zusätzlichen Positional Encodings werden verwendet. Beim Self Attention Layer gibt es nur einen Unterschied: Es kann sich nur auf frühere Positionen der Outputsequenz bezogen werden. Dieser Prozess wird auch als Future Masking bezeichnet. Außerdem ist noch zu erwähnen, dass der Decoder seine Keys und Values vom Encoder erhält.

Finale Linear & Softmax Layer

Da der Decoder einen Float-Vektor ausgibt, muss man diesen in ein Wort umwandeln. Das ist die Aufgabe des Linear und Softmax Layers. Der Linear Layer projiziert den Outputvektor des Decoders auf einen größeren sogenannten Logits Vektor. Die Länge des Vektors ist gleich des Vokabulars, das das Modell kennt. Jede Zelle des Vektors spiegelt den Score eines Wertes

wider. Der Softmax Layer wandelt diesen dann in eine Wahrscheinlichkeit um. Das Wort mit der höchsten Wahrscheinlichkeit wird dann als finalen Output gewählt.

Training

In dem ursprünglichen Forschungspapier wurde der Transformer durch Englisch-Deutsch und Englisch-Französisch Übersetzung trainiert. Der Output, also die Zielsprache in der Übersetzung, ist beim Input in den Decoder um eine Position nach rechts verschoben, da das Ziel des Models nicht ist den Decoder-Input zu kopieren, sondern das nächste Wort in der Übersetzung vorauszusagen. In der Loss-Funktion wird dann die echte Wahrscheinlichkeitsverteilung mit der vorausgesagten verglichen und die Gewichtungen bei der Backpropagation angepasst [39] [40] [41].

BERT

Das schon erwähnte BERT Modell baut auf den Ideen des Transformers auf, indem es mehrere Encoder nimmt und diese übereinanderstapelt (Base Model: 12 Encoder, Large Model: 24 Encoder). Dies unterscheidet BERT von vorherigen Sprachmodellen, bei denen eine Textsequenz nur von links nach rechts betrachtet wurde oder eine Betrachtung von links nach rechts und von rechts nach links kombiniert wurde. Letzteres wird als „shallowly bidirectional“ bezeichnet, da die Zusammenführung beider Betrachtungsrichtungen am Ende oder nur in bestimmten Teilabschnitten geschieht, wohingegen BERT natürlich bidirektional ist. Das Forschungspapier konnte zeigen, dass bidirektional trainierte Sprachmodelle wie BERT ein tieferes Verständnis von Sprache erreichen. Dieses schon vortrainierte Grundverständnis der Sprache kann dann für einen speziellen Anwendungsbereich verfeinert werden. Dieser Vorgang wird auch als Transfer-Learning bezeichnet. Mit BERT wird also ein allgemeines Sprachmodell bereitgestellt, weshalb es nun nicht mehr nötig ist für jeden Aufgabenbereich ein Modell von Grund auf aufzubauen.
BERT funktioniert folgendermaßen: Anders als bei unidirektionalen Modellen wird hier die ganze Textsequenz gleichzeitig gelesen. Daraufhin folgt die Berechnung und die gestapelten Self-Attention Layer ähnlich zu der Encoder Struktur. Beim Training werden zwei Methoden parallel angewandt: Zum einen wird eine neuartige Methode namens „Masked Language Modeling“ kurz MLM angewandt. Hierbei werden 15% der Wörter eines Satzes maskiert also verdeckt. Das Modell soll dann die maskierten Wörter erraten. Auch durch Next Sentence

Prediction (NSP) wurde BERT trainiert. Dabei wird dem Modell ein Paar von zwei Sätzen gegeben und es muss ermittelt werden, ob diese aufeinanderfolgende Sätze sind oder nicht. Das Ziel ist es den kombinierten Verlust beider Strategien zu minimieren. Zum Fine-Tuning für eine bestimmte Aufgabe wird ein zusätzlicher Layer hinter den Transformer gehangen. Beispielsweise ein Classification Layer, in Form von einem Fully-Connected und Softmax Layer, zur Textklassifizierung wie Sentiment Analysis. Das Modell kann aber auch zur Fragenbeantwortung trainiert werden. Dabei erhält BERT die Frage und markiert die Stelle im Text, die diese beantwortet [37] [43] [44] [45].

Nachteile von BERT

1. BERT wurde trainiert, indem maskierte Wörter erraten werden. Beim Fine-Tuning wird jedoch keine Maskierung verwendet, weshalb einfach nicht-maskierte Tokens als Output genommen werden. Es ist nicht klar, ob BERT eine korrekte Repräsentation für nicht maskierte Tokens erlangen kann.

2. Bei der Maskierung kann es vorkommen, dass voneinander abhängige Wörter gleichzeitig maskiert werden. BERT nimmt jedoch fälschlicherweise an, dass die zu erratenen Tokens unabhängig sind, was zu falschen Vorhersagungen führen kann. Dies wird an folgendem Beispiel deutlich:
In dem Satz „Ich fliege nach [MASK] [MASK]“ wären sowohl Los Angeles als auch New York korrekte Vorhersagen. Dadurch, dass Wörter für die Maskierungen gleichzeitig und unabhängig erraten werden, können auch Vorhersagen wie „Los York“ bei BERT als korrekt angesehen werden [46].

XLNet

Das 2019 erschienene XLNet hat bei vielen verschiedenen Aufgaben eine bessere Performance als BERT. Es baut auf den Erfolgen von BERT auf und behebt seine Schwächen mit dem Ziel Bidirektionalität zu erhalten und unabhängige Maskierung zu vermeiden. Hierbei muss aber zuerst der grundsätzliche Unterschied zwischen BERT und XLNet verdeutlicht werden. Bei BERT handelt es sich um ein Autoencoder Language Model. Diese Art von Sprachmodellen rekonstruiert originale Daten auf Basis von (z.B. durch Maskierung) korrumpierten Inputdaten. XLNet ist lediglich ein Feed-Forward Modell, welches das nächste Wort auf Basis von einem Satz an bekannten Wörtern vorhersagt. Auf den ersten Blick

erscheint diese Art von Vorhersage unidirektional: Man errät entweder das nächste oder das vorherige Wort. Bei XLNet wird das Modell jedoch gezwungen Bidirektionalität zu erlernen, indem die Wörter in zufälliger Reihenfolge erraten werden müssen. Diese Methode wird als Permutation Language Modeling bezeichnet [46] [49].

Transformer-XL

Zusätzlich dazu wird das Transformer XL Model verwendet. Transformer XL ist eine ebenfalls 2019 vorgestellte neuartige Architektur, die zwei Probleme von herkömmlichen Transformern zu beheben versucht:

1. Langzeitabhängigkeiten nicht modellierbar durch einfache Transformer
2. Kontextfragmentierung durch Nichtbeachtung von Satzbegrenzungen, was zu Optimierungsineffizienz, besonders bei kurzen Sätzen, bei denen Langzeitabhängigkeiten kein Problem darstellen, führt.

Transformer XL löst diese Probleme durch zwei Methoden: Segment-Level Recurrence Mechanism und Relative Positional Encoding. Bei Ersterem werden die berechneten Repräsentationen des vorherigen Segments im Cache gespeichert, um als erweiterten Kontext im nächsten Segment wieder benutzt zu werden. Um die Positional Encodings beim Zugriff auf gespeicherte Repräsentationen kohärent zu halten werden Relative Positional Encodings benutzt. Durch diese Änderungen an der Transformerarchitektur können 80% längere Abhängigkeiten als bei RNNs und 450% längere als bei traditionellen Transformern erlangt werden. Zudem ist diese Architektur mehr als 1800-mal schneller als traditionelle Transformer. Bei langen Sequenzen ist auch die Performance besser, da Langzeitabhängigkeiten modelliert werden können [47] [48].

Two-Stream Self-Attention

Zusätzlich zu den schon genannten Methoden wird auch die Maskierung beim Pre-Training abgewandelt: Bei traditionellen transfromerbasierten Sprachmodellen wird bei der Vorhersage von einem Token an der Stelle i sein Embedding vollständig maskiert. Dies ist besonders problematisch für Tokens, die am Anfang eines Satzes positioniert sind, da deren Wahrscheinlichkeitsverteilung anders ist als von Tokens an anderen Stellen. Um dies zu beheben, wird zusätzlich eine zweite Repräsentation genutzt, welchen nur die Positional Embeddings des gesuchten Tokens enthält. Diese Repräsentation wird Query Stream genannt. Für die Vorhersage des gesuchten Wortes wird dann sowohl der Query Stream als auch der Content Stream genutzt, welcher sowohl die Positional als auch die Token Embeddings aller vorheriger Tokens enthält [46] [49].

Honorable Mentions: ELECTRA

ELECTRA ist ein 2020 erschienenes Sprachmodell aus dem Hause Google und setzt seinen Fokus auf Effizienzsteigerung. Die Performancesteigerungen der letzten Jahre wurden durch neuartige Architekturen und Methoden aber auch durch größere Modelle mit mehr Parametern erreicht. Daraus folgen mehr Speicher- und Rechenaufwand und somit höhere Kosten. ELECTRA konnte nun erreichen, dass in gewissen Aufgabenbereichen ähnliche oder sogar bessere Resultate als BERT oder XLNet mit 75% weniger Rechenaufwand erzielt werden können. Darunter fallen GLUE (General Language Understanding Benchmark) und SQuAD (Question and Answering Benchmark). Sentiment Analysis fällt nicht unter diese Kategorie, weshalb ELECTRA für diese Projektarbeit nicht geeignet ist. ELECTRA sollte jedoch nicht unerwähnt bleiben, da es die Probleme der immer größer werdenden Modelle aufzeigt. Erreicht wird diese Effizienzsteigerung, indem das Pre-Training verändert wird. Es wird auch ein bidirektionales Modell basierend auf allen Inputs erlernt. Anstatt durch Maskierung den Input zu korrumpieren werden manche Tokens durch falsche, jedoch plausible, ersetzt. Inspiriert von GANs (General Adversarial Networks) soll dann zwischen echten und falschen Inputdaten unterschieden werden, indem klassifiziert wird welche Tokens ersetzt wurden und welche nicht. Dies wird für alle Tokens angewandt anstatt nur 15% wie bei der Maskierung von BERT. Dies hat zur Folge, dass ELECTRA weniger Beispiele für die gleiche Performance sehen muss. Das klassifizierende Neuronale Netzwerk wird Discriminator genannt und das „gegnerische“ neuronale Netzwerk, welches die Tokens ersetzt, heißt Generator [50].

Kapitel 4: Planung

In diesem Kapitel wird unter der Berücksichtigung der Erkenntnisse der vorherigen Kapitel dargelegt, welche Algorithmen und Techniken am besten im Kontext dieser Projektarbeit geeignet sind und wie diese eingesetzt werden können, um das Ziel der Projektarbeit zu erreichen.

4.1 Funktionale Eigenschaften

Auswahl einer Lernmethode

In Kapitel 3 wurden drei grundsätzliche Herangehensweisen vorgestellt, die ich nun bezüglich ihrer Eignung für diese Projektarbeit vergleiche. Die Diskussion welche Art des Lernens in Machine Learning am nützlichsten ist zieht sich durch alle Bereiche des maschinellen Lernens. Aber gerade im Bereich NLP konnte durch BERT ein Durchbruch erzielt werden, weshalb viele Experten (darunter auch das Unternehmen DeepMind und Facebooks Chief AI Scientist Yann LeCun) dies als Paradigmenwechsel in der Herangehensweise des Trainierens von neuronalen Netzwerken sehen [51]. Denn BERT und XLNet fallen weder unter die Kategorie Supervised Learning noch Unsupervised Learning. Es handelt sich hierbei viel mehr um Self-Supervised Learning. Diese Lernmethode hat das Ziel dateneffiziente KI zu ermöglichen. Während Unsupervised Learning und Reinforcement Learning nur sehr begrenzt Anwendung findet, konnte mit Supervised Learning bisher gute Performance erzielt werden. Jedoch wird diese Lernmethode durch die Menge an qualitativ aufgearbeiteten Datensätzen begrenzt. Dies stellt Stefan Seltz-Axmacher in einer Grafik sehr gut dar. Als Mitgründer von Starksy Robotics, eine Firma, die 2019 als erste einen autonomen Truck auf öffentlichen Autobahnen fahren lassen durfte, stellte er fest, dass er bei Supervised Machine Learning Systemen statt eines exponentiellen Anstieg an Performance einen exponentiellen Anstieg an Kosten und Zeit auffand [65]. Dies führt zu folgender Entwicklung der Performance:

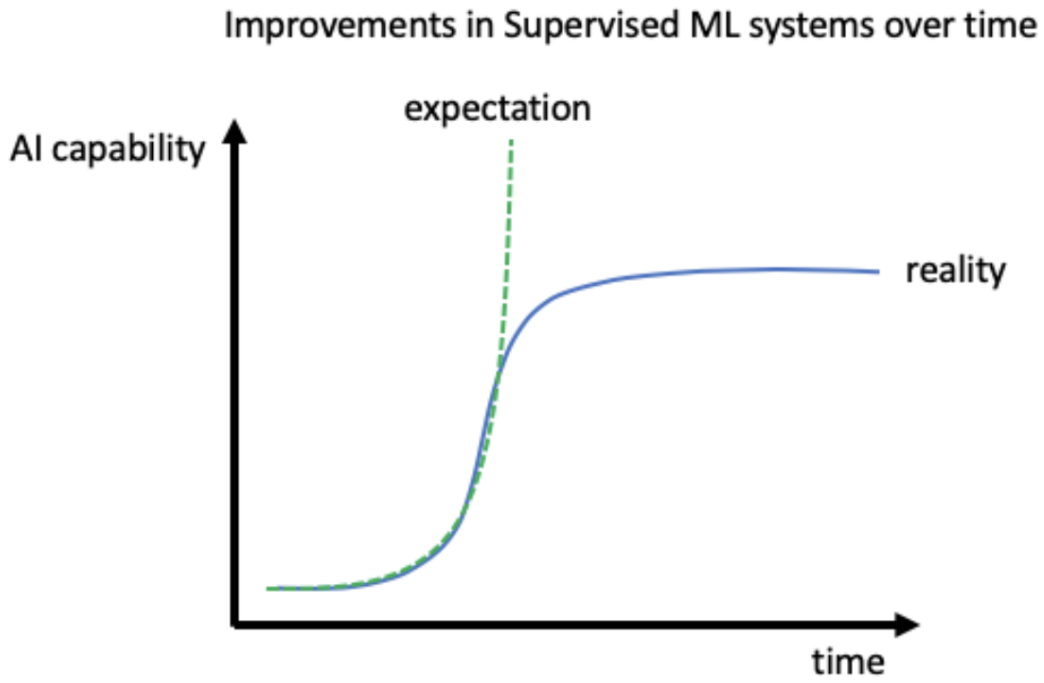


Abbildung 9: Kosten-Nutzen Graph eines Supervised ML Systems [65]

Zudem können neue Beispiele, die teilweise nur leicht von den Trainingsdaten abweichen, das neuronale Netzwerk dazu bringen, falsche Entscheidungen zu treffen. Deep Reinforcement Learning konnte zwar erstaunliche Performance in Spielen wie Starcraft 2, Dota 2 oder Go leisten. Jedoch ist diese Lernmethode nur auf einfache Probleme anwendbar. Außerdem wird viel Rechenleistung benötigt, was die Kosten solcher Projekte in die Höhe treibt. Diese Systeme sind darüber hinaus schlecht für Transfer Learning geeignet: Die Performance des Systems fällt stark ab, sobald es für ein anderes Spiel eingesetzt wird oder das Spiel, wofür es trainiert wurde, nur leicht verändert wurde. Self-Supervised Learning Systeme wie BERT und XLNet benötigen keine gelabelten Datensätze für das Pre-Training, den Hauptteil des Trainings, da diese bspw. durch Maskierung von dem System selbst gelabelt werden können. Dies macht diese Lernmethode dateneffizienter als Unsupervised Learning Ansätze und gleichzeitig weniger begrenzt durch Daten als Supervised Learning [52]. Hierbei sei jedoch angemerkt, dass das Lernen durch eigens erstellte Labels/Maskierung nur für das Lernen eines generellen Verständnisses von Sprache genutzt wird, während für das Fine-Tuning bspw. für eine Anwendung im Bereich Textklassifizierung weiterhin mit gelabelten Datensätzen gearbeitet wird, wovon jedoch weniger benötigt werden [53]. Wie schon in Kapitel 3 erläutert konnte diese Methode die Performance gegenüber früheren Modellen deutlich steigern. Auch ist anzumerken, dass im Bereich NLP diese Sprachmodelle allgemein sind und sie durch Transfer Learning für viele verschiedene Anwendungsgebiete

benutzt werden können. Aus den oben genannten Gründen glaubt Yann LeCun, Erfinder von CNNs, dass Self-Supervised Learning die Zukunft von künstlicher Intelligenz ist [52]. Auf Grund dieser Argumentation ist ein Self-Supervised System auch für meine Projektarbeit zu nutzen. Die bewiesene Performance gewährleistet ein leistungsfähiges Modell am Ende der Arbeit, die Dateneffizienz ermöglicht die Umsetzbarkeit, da die Erstellung und Nutzung großer Datensätze mit viel Aufwand verbunden ist und durch die aktuelle Relevanz dieser State-of-the-Art Methode sichert den Lernerfolg dieser Projektarbeit.

BERT vs XLNet

Bei der Vorstellung der beiden Modelle wird deutlich, dass beide die vorherige Generation von Modellen outperformt. Zum aktuellen Stand (Herbst 2020) scheint BERT besser erforscht zu sein als XLNet, da es mehr weiterführende Forschungen, mehr Erklärungen und Implementierungen gibt. Somit gibt es mehr Ressourcen, auf die man bei der Implementierung zurückgreifen kann. Aus den genannten Gründen wird im Folgenden auf Basis von BERT durch Fine-Tuning ein vollständiges Sprachmodell zur Sentiment Analysis von Wirtschaftsnachrichten entwickelt.

Parameter

In dem Forschungspapier, indem BERT vorgestellt wurde, werden für das Fine-Tuning verschiedene Parameter empfohlen. Als Batch Size soll 16 oder 32 genommen werden. Für den Optimizer Adam soll eine Learning Rate zwischen 5e-5 und 2e-5 gewählt werden. Trainiert soll 2 bis 4 Epochen lang [37]. Bei dem Fine-Tuning von BERT in dieser Projektarbeit sollte sich an diesen empfohlenen Parametern orientiert werden. Hierbei sollte jedoch zum einen die Umsetzbarkeit beachtet werden und zum anderen überprüft werden, ob und in welcher Kombination diese Parameter tatsächlich optimal sind.

Datensatz

Die Daten, die zum Trainieren von Machine Learning Modellen genutzt werden, sind ausschlaggebend dafür in welchem Bereich und wie gut das Modell performt. So sind die Beschränkungen des Datensatzes, zumindest teilweise, gleichzeitig auch die Beschränkungen des Modells. Dies lässt sich an dem genannten Beispiel des Reinforcement Learning

feststellen, bei dem das Modell Experte in einem Spiel ist, jedoch bei einem anderen Spiel nur eine schlechte Leistung erbringt. Diese Art von Machine Learning Modellen, die sich auf eine bestimmte Aufgabe spezialisiert, wird Narrow AI genannt und findet heutzutage in vielen Bereichen ihre Anwendung [66]. Der Datensatz bzw. das Gebiet, auf dem das Modell trainiert wird, ist also entscheidend für die Nutzbarkeit des Modells auf dem gewünschten Anwendungsgebiet.

Das Ziel dieser Projektarbeit ist es ein Machine Learning Modell zu entwickeln, welches in dem Anwendungsgebiet Wirtschaft die Nachrichtenüberschriften nach ihrem Sentiment korrekt klassifizieren kann. Dies ist zwar ein wohl definierter Bereich der Nachrichten, jedoch auch ein sehr breiter. Oft können Nachrichten aus dem Weltgeschehen, welches auch die Wirtschaft beeinflusst, die Nachrichten dominieren wie bspw. Covid-19. Auch Nachrichten über Quartalsberichte, neue Produkte oder Projekte können vorkommen. Aber auch Berichte darüber wie sich der Markt entwickelt, ob er steigt oder fällt. Zudem könnte auch die Länge der Sätze, die im Datensatz enthalten sind, das Modell beeinflussen. Längere Sätze enthalten mehr Informationen, aus denen BERT das Sentiment schließen kann, diese Informationen können jedoch auch verwirrend sein, wenn mehrere Sentiments in einem Satz enthalten sind. Des Weiteren muss für BERT eine feste Länge der Inputsequenz festgesetzt werden. Sind Sätze zu lang, werden diese gekürzt. Eine große Inputlänge führt aber auch zu mehr Speicheraufwand beim Trainieren. Wichtig ist auch wie die Zusammensetzung aus positiven, negativen und neutralen Nachrichten in dem gesamten Datensatz ist. Überwiegen bspw. positive Nachrichten im Datensatz, so kann es dem Modell schwerer fallen zu lernen, wie neutrale und negative Nachrichten korrekt zu klassifizieren sind. Weiterhin ist zu beachten, dass unterschiedliche Nachrichtenagenturen einen anderen Fokus haben. Hierzu ein Beispiel der Schlagzeilen vom 22.11.20 von dem Wall Street Journal, Barron's und Bloomberg:

„Judge Dismisses Trump Campaign Suit in Pennsylvania "-Wall Street Journal

"The World is Full of Opportunities. Put these 20 International Stocks on Your Radar"-Barron's

"Astra-Oxford Shot is Key to Escaping Pandemic for Many Nations"-Bloomberg

Hierbei wird deutlich, dass bei einer Nachricht Politik, bei der anderen ein Gegenmittel für die Pandemie und bei der anderen eine Kaufempfehlung für bestimmte Aktien im Vordergrund steht. Deshalb ist dies auch bei dem Datensatz zu beachten. Wenn allgemein Wirtschaftsnachrichten klassifiziert werden sollen, dann müssen auch verschiedene Themen und Nachrichtenagenturen im Datensatz enthalten sein. Außerdem existieren zum einen

Nachrichten, die allgemein für die Wirtschaft oder einen bestimmten Sektor sind, und zum anderen firmenspezifische Nachrichten. Es ist also nicht nur wichtig die verschiedenen Themen der allgemeinen Wirtschaft zu umfassen, sondern auch die Themen unterschiedlicher Firmen. Bei einem Immobilienunternehmen tauchen also eher Themen wie die Schließung oder Eröffnung von Geschäften auf, während bei einem Technologieunternehmen Themen wie Cloud oder Künstliche Intelligenz in den Vordergrund rücken. Aus diesem Grund ist bei dem Datensatz auf eine Mischung der verschiedenen Nachrichtenarten zu achten, wobei Nachrichten zu einzelnen Unternehmen nur bedingt im Datensatz darstellbar sind, da der Aufwand zu groß wäre alle Unternehmen im Datensatz zu enthalten. Aus diesem Grund ist eine Auswahl von Unternehmen nötig, in der verschiedene Unternehmen aus verschiedenen Branchen vorkommen. Insgesamt werden in dieser Projektarbeit 23 Unternehmen dafür verwendet. Im Folgenden dazu ein Ausschnitt aus der Liste: Apple, Disney, Intel, Johnson & Johnson, Coca-Cola, Simon Property Group, Exxon Mobil, AT&T, Unilever, Waste Management und Lockheed Martin. Durch den Trade-Off zwischen Anzahl an inkludierten Unternehmen und Aufwand bei der Erstellung des Datensatzes, limitiert, wie auch in diesem Abschnitt beschrieben, der Datensatz und so auch die Auswahl der Unternehmen die Performance des Modells. Auch das Herkunftsland des Unternehmens könnte eine Rolle spielen, falls BERT Begriffe aus dem US-amerikanischem Raum besser versteht als von anderen Regionen. Es wurde versucht viele verschiedene Unternehmen auszuwählen, jedoch kann eine allumfassende Abdeckung nur bei einem sehr großen Datensatz gelingen.
Außerdem möchte ich noch auf zwei weitere Dinge hinweisen. Zum einen ist es bei Wirtschaftsnachrichten oft der Fall, dass das Sentiment nicht genau bestimmbar ist. Dies wird an der Überschrift aus dem Wall Street Journal deutlich. In dem Satz „Judge Dismisses Trump Campaign Suit in Pennsylvania“ könnte man zu verschiedenen Schlussfolgerungen bei der Ermittlung des Sentiments kommen: Ein naiver Ansatz würde zeigen, dass durch das negative Wort „dismiss“ ein negatives Sentiment folgt. Jedoch handelt es sich hier eher um ein Negationswort als ein negatives Wort, weshalb auch der Kontext eine Rolle spielt. Ist es nun also bezüglich des Sentiments positiv oder negativ, dass die Klage Trumps abgelehnt wurde? Nun, genau dies ist nicht eindeutig bestimmbar und hängt von der Perspektive der Person ab, die das Sentiment bestimmen will. Eine Möglichkeit wäre den Satz als neutral zu labeln. Jedoch sind neutral und unbestimmbar unterschiedliche Dinge. Dies führt mich zu meinem zweiten Punkt: Je nach Neigung der Person kann es zu einer Parteilichkeit im Datensatz führen. Dies ist auch recht gut erforscht. Die Parteilichkeit im Datensatz, welche bspw. durch Unterrepräsentation im Datensatz verursacht werden kann, kann von dem Modell

geerbt werden, was zu (oft unglücklichen) falschen Klassifizierungen führen kann [67]. Eine weitere Limitation ist, dass BERT nur der englischen Sprache mächtig ist und somit nur englische Daten verwendet werden können. Dadurch sind die möglichen Daten und Sichtweisen, ähnlich wie bei den verschiedenen Nachrichtenagenturen, begrenzt.

Es werden zum Erreichen des Zieles der Projektarbeit zwei Datensätze genutzt: Ein eigens erstellter und einer von der Webseite Kaggle. So kann untersucht werden, wie BERT mit verschiedenen Daten umgeht, da die Quellen der Datensätze unterschiedlich sind. Und zum anderen kann untersucht werden, wie unterschiedlich große Datensätze sich auf die Performance von BERT auswirken. Für die Erstellung eines eigenen Datensatzes ist die Entwicklung eines Webscrapers nötig, welcher Nachrichtenüberschriften von der Finanzseite „finviz.com“ holt und diese dann in eine passende CSV (Comma Seperated Values) Datei schreibt. Finviz wird hierbei genommen, da es eine Vielzahl von unterschiedlichen Nachrichten enthält. Alternativen wären Yahoo Finance oder SeekingAlpha gewesen. Ersteres enthält zwar auch Nachrichten aus verschiedenen Quellen, jedoch hauptsächlich von Yahoo. Und SeekingAlpha bezieht sich zwar auf andere Nachrichtenagenturen oder Presseerklärungen, verfasst die Artikel sowie die Überschriften jedoch selbst. Um eine Parteilichkeit oder ein Überwiegen einer Presseagentur zu vermeiden wird Finviz als Quelle genommen. Hier werden nämlich keine Nachrichten verfasst, sondern lediglich Nachrichten anderer präsentiert. Des Weiteren werden diese nicht nach Popularität o.ä. sortiert, sondern lediglich zeitlich geordnet. Dies sollte zu einer Minimierung der Parteilichkeit aus dieser Quelle führen. Nach der Beschaffung der Daten werden diese anschließend durch ein eigens entwickeltes Annotierungsprogramm manuell die Labels, also das Sentiments, für die Sätze erstellt. Die Nachrichten enthalten sowohl allgemeine Nachrichten zu Politik und Wirtschaft als auch spezifische zu Unternehmen, sind aber stets im Kontext von Finanz und Wirtschaft. Angedacht ist eine Größe des Datensatzes von 8000 Sätzen. Da Nachrichten, anders als Produktbewertungen oder Kommentare, von Natur aus oft neutral sind, ist es hier sinnvoll zwischen drei Sentiments zu unterscheiden (positiv, negativ, neutral), um die neutralen Nachrichten nicht falsch zu klassifizieren. Bei diesem Datensatz, so wie bei allen genutzten, gelten stets die bereits beschriebenen Bemerkungen und Limitierungen. Zusätzlich ist anzumerken, dass der Sammelzeitraum von den Daten eine Rolle spielt. Während die Daten gesammelt wurden spielten besonders Covid-19 und die daraus resultierende Wirtschaftskrise sowie der zu der Zeit amtierende US-Präsident Donald Trump eine große Rolle. Das Wort „Vaccine“ kommt 324 mal, „Coronavirus“ 313 mal, „Covid-19“ 275 mal und „Trump“ 266 mal vor. Hierbei ist zum einen eine Parteilichkeit gegenüber diesen Begriffen möglich. Und

zum anderen hat BERT die Möglichkeit gerade solche Sätze mit diesen Begriffen besonders gut zu klassifizieren, da die entsprechende Datenmenge vorhanden ist. Den Namen des US-Präsidenten für die Periode 2021-2025 ist dem Modell jedoch unbekannt. Auch ist zu bemerken, dass der Datensatz und somit das Modell nur die Krisenzeit kennt. Dies hat den Vorteil, dass negative, positive und neutrale Nachrichten bezüglich ihrer Häufigkeit recht ausgeglichen sind (wird in den folgenden Abschnitten dargelegt). Dies wiederum könnte dazu führen, dass dadurch das der Datensatz recht ausgeglichen ist die Performance optimiert wird oder das in Zeiten des wirtschaftlichen Aufschwungs die Performance des Modells leidet, da solche Zeiten ihm unbekannt sind.

Der Datensatz von Kaggle namens „Phrase Bank“ enthält 4837 Überschriften von Wirtschaftsnachrichten und wurde erstmals in einem Forschungspapier aus dem Jahre 2014 verwendet. Auf Kaggle veröffentlicht wurde es im April 2020. Das Sentiment wird in drei Klassen unterteilt: positiv, negativ und neutral. Es wurde von 16 Annotierern mit adäquatem Hintergrundwissen über die Finanzmärkte erstellt. Zudem wurde jeder Satz von mehreren Personen annotiert, wodurch gezeigt wurde, dass das Sentiment von Wirtschaftsnachrichten nicht immer eindeutig bestimmbar ist bzw. dass das Sentiment je nach Meinung sich unterscheiden kann. Die Parteilichkeit sollte dadurch jedoch verringert worden sein, da die Meinung der Mehrheit als Label genommen wurde [63] [64].

Beim Betrachten der beiden Datensätze sind einige Dinge auffallend. Zum einen ist der Finviz Datensatz fast doppelt so groß wie der von Kaggle. Zum anderen ist die Verteilung der Sentiments anders. Beim Finviz Datensatz: Positiv = 3474 (43%), negativ = 2608 (33%), neutral = 1918 (24%). Bei dem Phrase Bank Datensatz ist die Verteilung wie folgt: Positiv = 1363 (28%), negativ = 604 (12%) und neutral = 2879 (60%). Hierbei wird deutlich, dass der Datensatz von Kaggle unausgeglichener ist. Jedoch ergänzen sich beide Datensätze bezüglich ihrer Verteilung der Klassen gut. Wie erwartet hat der Finviz Datensatz einen hohen Anteil an negativen Nachrichten. Weiterhin überwiegen dort jedoch die positiven Nachrichten. Beim Phrase Bank Datensatz überwiegen neutrale Nachrichten, da die Sätze, die dort enthalten sind, recht verschieden von meinem eigenen Datensatz sind. Dies wird an folgenden Beispielen deutlich:

„Operating profit totaled EUR 3.8 mn , down from EUR 4.5 mn in the corresponding period in 2005 “

“In the Baltic countries , sales fell by 42.6 %”

"Operating profit for the 12-month period decreased from EUR2 .9 m while turnover increased from EUR24 .5 m , as compared to the financial year 2004"

Solche Finanzreporte stellen, wie am Anteil der neutralen Nachrichten zu erkennen, den Großteil der Nachrichten im Datensatz dar. Ähnliche Neuigkeiten, wie bspw. Quartalsberichte, sind zwar auch in meinem enthalten, jedoch weniger häufig. Des Weiteren fällt auf, dass es sich hier um Nachrichten im europäischen Raum handelt. Das hat den Grund, dass der Datensatz an der finnischen Aalto Universität erstellt wurde und die Wissenschaftler europäische Nachrichten ihrem Datensatz zu Grunde gelegt haben. An den Jahreszahlen, die in den Sätzen des Datensatzes erwähnt werden, lässt sich auch sagen, dass die Daten ca. von den Jahren 2004-2011 sind. Dadurch sind Nachrichten aus Zeiten des wirtschaftlichen Booms, Abschwung und Erholung enthalten. Außerdem ist zu erwähnen, dass im Phrase Bank Datensatz deutlich mehr lange Sätze enthalten sind. Das lässt sich daran erkennen, der Phrase Bank Datensatz größer ist als der von Finviz: 657KB im Vergleich zu 548KB obwohl er weniger Sätze enthält. Abschließend sollte gesagt sein, dass in den genannten Punkten die beiden Datensätze sich recht unterschiedlich sind und sich so auch ergänzen können. Dies und wie sich die Datensätze auf die Performance von BERT auswirken soll in Experimenten untersucht werden.

Aspect vs Sentence Sentiment Analysis

Wie in Kapitel 3 dargestellt kann die Sentiment Analysis auf drei Eben durchgeführt werden: Document, Sentence oder Aspect Ebene. Da in dieser Projektarbeit Nachrichtenüberschriften, und somit nur Sätze, auf ihr Sentiment untersucht werden fällt die Document Sentiment Analysis weg bzw. ist mit der Sentence Sentiment Analysis gleichzusetzen. Für die Wahl einer Aspect Sentiment Analysis würde eine möglicherweise erhöhte Genauigkeit sprechen. Jedoch wird dieser Vorteil dadurch deutlich relativiert, dass auf bekannten Finanzseiten wie „finviz", „Seeking Alpha" oder „Yahoo! Finance" die Nachrichten schon nach den jeweiligen Unternehmen sortiert sind. Für Nachrichten, in denen mehrere Unternehmen genannt werden, könnte die Aspect Sentiment Analysis zwar dennoch höhere Genauigkeit bieten. Dieser Vorteil ist jedoch mit der erhöhten Komplexität abzuwiegen. Aus den genannten Gründen und weil diese Projektarbeit nicht die Bewertung von Aspect Sentiment Analysis mit BERT als hauptsächliche Zielsetzung besitzt wird hier auf diese verzichtet.

4.2 Nicht-funktionale Eigenschaften

Auswahl der Programmiersprache

Für den Bereich Machine Learning sind mehrere Programmiersprachen geeignet. Die gängigsten sind Python, C++, Java, JavaScript, und R. Wobei 60% der Machine Learning Entwickler Python nutzen. Python gilt als einfach zu lernen, leicht skalierbar und hat viele mächtige Bibliotheken wie „pandas“, „sklearn“ oder „matplotlib“. Andere Sprachen wie C++ haben jedoch andere Vorteile wie bspw. eine höhere Effizienz [54]. Für diese Projektarbeit entscheide ich mich dennoch für die Sprache Python, da ich dort bisher am meisten Erfahrung sammeln konnte und Quellen sowie benötigte Bibliotheken leicht zugänglich sind.

Auswahl Entwicklungsumgebung

Bei der Programmierung im Bereich Machine Learning mit Python gibt es zwei grundsätzliche Möglichkeiten: Zum einen kann eine IDE wie PyCharm genutzt werden, um ein Skript zu schreiben. Dies kann auch mit Konzepten der Objektorientierung geschehen. Zum anderen ist es möglich ein sogenanntes Notebook, wie z.B. Jupyter Notebook, zu nutzen. Hierbei können Code und Textteile in Zellen unterteilt werden, welche separat ausgeführt werden können. Dies erleichtert es zu experimentieren und Codeblöcke zu testen. Besonders gut ist Jupyter Notebook auch für die Datenvisualisierung, da die Ausgabe sofort angezeigt wird. Auf der anderen Seite existiert die Gefahr, dass das Notebook schnell zu einem Monolith wird, welcher nur sehr schwer versioniert und gewartet werden kann. Für kleinere Forschungen wie diese Projektarbeit überwiegen jedoch die Vorteile, weshalb die Nutzung von Jupyter Notebook sinnvoll erscheint. Hierzu stellt Google die Plattform „Google Colab“ bereit auf der kostenlos Rechenleistung bezogen werden für die Ausführung von Notebooks. Dies bringt viele Vorteile, da die bereitgestellten High End Grafikkarten für das Trainieren des neuronalen Netzwerks genutzt werden können. Dies ist auch oft nötig, da das Trainieren nicht nur Rechenleistung, sondern auch viel Grafikspeicher benötigt, weshalb im Rahmen dieser Projektarbeit diese Möglichkeit genutzt wird.

Bibliotheken

Im Folgenden stelle ich die Bibliotheken vor, die in dieser Projektarbeit genutzt werden. Der Webscraper benötigt die Bibliothek „requests“, um HTTP Anfragen zu verschicken. Dadurch erhält das Programm die Webseite als Text [55]. „beautifulsoup“ ist der beliebteste Parser für XML und HTML Formate. Durch diese Bibliothek ist es dann also möglich die relevanten Daten, also die Nachrichten, über die HTML-Tags zu holen [56]. Die Bibliothek „re“ ermöglicht es mit regulären Ausdrücken die URL der Hauptseite von finviz.com mit URLs für bestimmte Unternehmen zu unterscheiden [57]. Dies ist notwendig, da je nach Seite die Daten über andere HTML-Tags geholt werden müssen. „pathlib“ bietet Methoden für Dateipfade. So kann z.B. abgefragt werden, ob eine Datei existiert bevor sie gelesen wird. Eine besonders wichtige Bibliothek ist hingegen „pandas“. Diese stellt effiziente DataFrame Objekte zur Verfügung. Zudem ermöglicht die Bibliothek CSV, Textdateien und weitere Dateiformate zu schreiben und zu lesen. Es ist auch möglich DataFrame Objekte zu manipulieren bspw. mit Funktionen, die Duplikate entfernen [58].

Für den Machine Learning Teil dieser Projektarbeit sind drei Bibliotheken notwendig. Bei der Wahl eine Machine Learning Frameworks gibt es zwei Möglichkeiten: Das von Facebook entwickelte Pytorch oder Tensorflow von Google. Die allgemeine Meinung ist, dass Tensorflow besser skaliert und besser geeignet für produktionsreife Modelle ist. PyTroch hingegen hat eine intuitivere Syntax, was Vorteile in dem Bau von Prototypen und in der Forschung bringt, weshalb mittlerweile mehr Forschungspapiere mit PyTorch veröffentlicht werden [59] [60]. Essenziell ist auch die Bibliothek „transformers“, welche Architekturen wie BERT für NLP zur Verfügung stellt. Hiermit können diese Pre-Trained Modelle geladen, angewandt und evaluiert werden. Sowohl das Fine-Tuning als auch das Trainieren von Grund auf wird ermöglicht [61]. Auch „scikit-learn“ wird in dieser Projektarbeit genutzt. Hauptsächlich, um den gelabelten Datensatz in einen Trainingsteil und einen Testteil zu splitten.

Kapitel 5: Umsetzung

In diesem Kapitel wird die Umsetzung der Projektarbeit erläutert. Das Kapitel umfasst den Machine Learning Algorithmus, den Labeler für den Datensatz und den Webscraper.

5.1 Webscraper

In dem Programm werden drei Methoden definiert und ein Skript, welches diese benutzt, um Nachrichtenüberschriften aus dem Web zu sammeln und diese in einer CSV Datei abzuspeichern bzw. anzuhängen. Als Datenquelle wird die Webseite finviz.com benutzt. Da Webscraper auf den HTML Code der jeweiligen Webseite spezialisiert sind, muss der Code für andere Webseiten erweitert oder bei Veränderungen der Datenquelle aktualisiert werden.

Get_headlines

```
#expects url for finviz news site or for finviz news of a ticker symbol
#returns list of headlines

def get_headlines(url):
  pattern = r"(https://finviz.com/quote.ashx\?t=.*)"
  agent = {"User-Agent":"Mozilla/5.0"}
  page = requests.get(url, headers = agent).text
  bs = BeautifulSoup(page)

  #check if url is finviz main news site
  if url == "https://finviz.com/news.ashx":
    headlines = bs.find_all("a", class_="nn-tab-link")
    #delete all non-news items
    del headlines[91:]
    del headlines[0]
    return headlines

  #check if url is news of ticker symbol on finviz
  elif re.match(pattern, url):
    headlines = bs.find_all("a", class_="tab-link-news")
    return headlines

  #unexpected url
  else:
    print("This URL is not supported!")
    return None
```

Diese Methode erhält eine URL, welche darauf geprüft wird, ob sie die allgemeine oder firmenspezifische Nachrichtenseite von finviz.com ist. Falls dies der Fall ist werden die Nachrichtenüberschriften über die Klasse der HTML-Tags in eine Liste geschrieben, welche dann zurückgegeben wird. Die HTML-Tags sind auf den firmenspezifischen Seiten eindeutig auf die Nachrichtenüberschriften zurückzuführen. Auf der Hauptnachrichtenseite gehören jedoch auch andere Inhalte dieser HTML-Klasse an, weshalb diese aus der Liste zuerst entfernt werden. Im Falle, dass eine unbekannte URL in die Methode übergeben wird, wird auf der Konsole ausgegeben, dass diese URL nicht unterstützt wird.

Headlines_to_df

Diese Methode erhält eine Liste von Nachrichtenüberschriften und wandelt diese in das DataFrame-Format von der Bibliothek Pandas um. Hierzu wird ein Dataframe Objekt erstellt mit den Spaltenüberschriften „news“ und „sentiment“. Die News werden als die übergeben Liste definiert und das Sentiment als 0, welches im Labelingprozess korrigiert wird.

Append_df

```
#expects data frame and file name as input
#appends and saves data frame to unlabeled dataset
def append_df(news_sentiment_df, file_name):
  my_file = Path(file_name)
  if my_file.exists():
    print("Appending to existing file named " + file_name)
    orig_df = pd.read_csv(file_name)
    new_df = pd.concat([orig_df, news_sentiment_df], ignore_index=True).drop_duplicates()
    new_df.to_csv(file_name, index=False, header = True, encoding='utf-8-sig')
  else:
    print("Creating new file named" + file_name)
    news_sentiment_df.to_csv(file_name, index=False, header = True, encoding='utf-8-sig')
```

Dieser Methode wird ein Dataframe Objekt und ein Dateiname übergeben. Falls die Datei bereits existiert wird sie auch als Dataframe Objekt geladen und mit dem übergebenen Dataframe konkateniert. Hierbei werden Duplikate verhindert. Anschließend wird der resultierende Dataframe als Datei gespeichert. Andernfalls wird der übergeben Dataframe direkt als Datei gespeichert.

Hauptprogramm

Hier wird eine Liste von Finviz-URLs erstellt bestehend aus der Hauptseite und Firmenseiten, welche über das Tickersymbol aufgerufen werden können. Es werden dann mit get_headlines die Nachrichtenüberschriften jeder Seite geholt und mit headlines_to_df sowie append_df konkateniert und in eine Datei abgespeichert.

5.2 Sentiment Labeler

In diesem Programm werden zwei Methoden und zwei Skripte definiert. Das Ziel ist es einen ungelabelten Datensatz zu durchlaufen, dem Nutzer jeden Satz zu zeigen, sodass dieser das Sentiment bestimmen kann. Diese Eingabe wird von dem Programm entgegengenommen und die Datensätze entsprechend aktualisiert.

Append_df

Diese Methode ist ähnlich zu der des Webscrapers. Der Unterschied ist hierbei, dass zum Überschreiben des Datensatzes die Methode update_csv genutzt wird, da diese Funktion mehrmals im Programm genutzt wird.

Update_csv

Hier wird lediglich der übergeben Dataframe in die übergebene Datei geschrieben bzw. überschrieben.

Labeling-Prozess

```
#unlabeled dataset
file_name = "news_headlines.csv"
#labeled dataset
new_file = "news_headlines_sentiment.csv"
news_sentiment_df = pd.DataFrame(columns=["news", "sentiment"])
my_file = Path(file_name)
if my_file.exists():
  #load and go through unlabeled dataset
  df = pd.read_csv(file_name, encoding='utf-8-sig', error_bad_lines=False)
  print("Loaded " + file_name)
  for index, row in df.iterrows():

    user_input = -1
    range = [0, 1, 2]
    #ask for user input until acceptable number is entered
    while user_input not in range:
      print("##########################################################################")
      print(row["news"])
      try:
        user_input = int(input("Positive: 0\nNegative: 1\nNeutral: 2\n"))
      except ValueError as err:
        print("\nPlease enter an Integer!\n")
        pass
    new_element = 0
    #prepare labeled row according to input
    if user_input == 0:
      new_element = [row["news"], 0]
    elif user_input == 1:
      new_element = [row["news"], 1]
    elif user_input == 2:
      new_element = [row["news"], 2]

    #save labeled sentence to labeled dataset
    news_sentiment_df.loc[len(news_sentiment_df)] = new_element
    append_df(news_sentiment_df, new_file)

    #delete sentence from unlabeled dataset
    index_name = df[df["news"] == row["news"]].index
    df.drop(index_name, inplace=True)
    update_csv(df, file_name)

else:
  print("File not Found")
```

Hier wird der ungelabelte Datensatz, sofern existent, als Dataframe geladen und Satz für Satz durchlaufen. Der Benutzer gibt das Sentiment ein. Dabei kann er zwischen 0 für positiv, 1 für negativ und 2 für neutral unterscheiden. Falsche Eingaben werden abgefangen. Das Sentiment mit dem dazugehörigen Satz wird dem Dataframe des gelabelten Datensatzes hinzugefügt und als Datei gespeichert. Der Satz wird zudem aus dem ungelabelten Datensatz gelöscht.

Eine Ausgabe könnte wie folgt aussehen:

```
Loaded news_headlines.csv
######################################################################
Comcast Stock Climbs As Activist Investor Trian Fund Maneuvers For Changes
Positive: 0
Negative: 1
Neutral: 2
0
Appending to existing file named news_headlines_sentiment.csv
Old Data Frame:
                                                   news  sentiment
0     UPDATE 3-Brazil economy back to 2009 size afte...          0
1     GLOBAL MARKETS-Manufacturing data lifts stocks...          1
2     TREASURIES-Yields move higher after U.S. manuf...          2
3     UPDATE 2-Dollar weakness lifts pound to 8-mont...          2
4     UPDATE 1-U.S. House Oversight Committee to sub...          0
...                                                 ...        ...
7995  Trian Investment in Comcast Fuels Debate on Br...          0
7996                              Is Roku Stock a Buy?           1
7997                10 Most Profitable TV Shows in 2020          2
7998  Comcasts Amy Banse Transitions to Senior Advis...          1
7999  Comcast and REVOLT Sign Agreement to Expand th...          2

[8000 rows x 2 columns]
New Data Frame:
                                                   news sentiment
0     UPDATE 3-Brazil economy back to 2009 size afte...         0
1     GLOBAL MARKETS-Manufacturing data lifts stocks...         1
2     TREASURIES-Yields move higher after U.S. manuf...         2
3     UPDATE 2-Dollar weakness lifts pound to 8-mont...         2
4     UPDATE 1-U.S. House Oversight Committee to sub...         0
...                                                 ...       ...
7996                              Is Roku Stock a Buy?          1
7997                10 Most Profitable TV Shows in 2020         2
7998  Comcasts Amy Banse Transitions to Senior Advis...         1
7999  Comcast and REVOLT Sign Agreement to Expand th...         2
8000  Comcast Stock Climbs As Activist Investor Tria...         0

[8001 rows x 2 columns]
Overwriting news_headlines_sentiment.csv
Overwriting news_headlines.csv
######################################################################
```

Vermeidung von Duplikaten

```
#removes duplicates preemptively to avoid labeling already labeled data
#duplicates may occur if already labeled sentences are added again to the unlabeled dataset during workflow
print("Deleting duplicates")

#unlabeled dataset
file_name = "news_headlines.csv"
#labeled dataset
new_file_name = "news_headlines_sentiment.csv"
news_sentiment_df = pd.DataFrame(columns=["news", "sentiment"])
orig_file = Path(file_name)
new_file = Path(new_file_name)
if orig_file.exists() and new_file.exists():
  #read unlabeled dataset. erroneous rows are skipped to increase robustness of the labeling process
  df = pd.read_csv(file_name, encoding='utf-8-sig', error_bad_lines=False)
  print("Loaded " + file_name)
  #go through the unlabeled dataset
  for index, row in df.iterrows():

    new_element = [row["news"], 3]
    #save labeled sentence to new file
    #create data frame only containing the current row
    if len(news_sentiment_df)!=0:
      news_sentiment_df.iloc[0] = new_element
    else:
      news_sentiment_df.loc[len(news_sentiment_df)] = new_element
    #read labeled dataset as data frame
    orig_df = pd.read_csv(new_file)
    #concat labeled dataset and current row of unlabeled dataset and drop duplicates
    new_df = pd.concat([orig_df, news_sentiment_df], ignore_index=True).drop_duplicates(subset=['news'], keep='last')

    #if sizes are same the sentence was already in the labeled dataset => delete from unlabeled dataset and update csv
    if (orig_df.size == new_df.size):
      index_name = df[df["news"] == row["news"]].index
      df.drop(index_name, inplace=True)
      update_csv(df, file_name)
      print("Duplicate removed")

else:
  print("File not Found")
```

Vor dem Labeling-Prozess wird dieser Teil des Programms durchlaufen, um zu verhindern, dass im gelabelten Datensatz vorhandene Sätze nochmals zum Labeling angezeigt wird. Hierzu wird der ungelabelte Datensatz durchlaufen. Satz für Satz wird dem gelabelten Datensatz hinzugefügt und Duplikate basierend auf dem Text des Satzes entfernt. Falls die Größe des gelabelten Datensatzes vor und nach diesem Prozess gleich ist, so handelt es sich bei dem aktuellen Satz um einen schon gelabelten sowie vorhandenen Satz im gelabelten Datensatz, weshalb dieser nun schon vor dem Labeling-Prozess entfernt wird.

5.3 Sentiment Classifier

Im Folgenden werde ich den Hauptteil, also das Fine-Tuning von BERT, erläutern. Der grundsätzliche Ablauf des Programms gestaltet sich wie folgt: Zuerst werden beide Datensätze geladen, welche zusätzlich auch zusammengeführt werden, um später die Performance vergleichen zu können. Anschließend folgt das Data Pre-processing, bei dem die Daten in ein BERT-freundliches Format gebracht werden. Die Daten werden unterteilt in

Trainings- und Testtteil. Schlussendlich werden die Parameter, wie z.B. learning rate, und mit diesen dann das Training gestartet. Nach jedem Durchlauf des Datensatzes (Epoch) wird die Performance auf dem Model unbekannte Daten getestet (Test Set).

Laden der Datensätze

```
# loading phrase bank dataset and correcting format

phrase_bank_dataset = "all-data.csv"
phrase_bank_dataset_file = Path(phrase_bank_dataset)
file_loaded = False
while not file_loaded:
  if phrase_bank_dataset_file.exists():
    phrase_dataset = pd.read_csv(phrase_bank_dataset, encoding='latin-1', names=["sentiment", "news"])
    phrase_dataset = phrase_dataset[["news", "sentiment"]]
    phrase_dataset["sentiment"].replace(['positive', 'negative', 'neutral'], [0,1,2], inplace=True)
    file_loaded = True
    print("Dataset Loaded")
  else:
    print("File not Found")
print(phrase_dataset)
```

Das Laden des Phrase Bank Datensatzes, wie hier gezeigt, ist ähnlich zu dem Laden des Finviz Datensatzes. Es müssen jedoch hier zusätzlich die Labels korrigiert und Spaltenüberschriften hinzugefügt werden. Falls ja wird die Datei in ein Data Frame Objekt geladen. Zuerst wurde hierbei versucht mit for-Schleifen durch den Datensatz zu laufen, um die Labels in das korrekte Format zu bringen. Dies stellte sich jedoch als sehr rechenaufwändig heraus, weshalb nun das gleiche Ergebnis über die „replace“ Methode von Pandas gelöst wird. Die Performance konnte dadurch deutlich verbessert werden. Vor dem Laden wird zudem überprüft, ob die Datei vorhanden ist.

NewsSentimentDataset Klasse

Der Code dieser Klasse ist von der huggingface Bibliothek. Die Klasse wird hier nur erwähnt, um darauf hinzuweisen warum diese Klasse notwendig ist: Beim Training wäre es zu speicheraufwändig den gesamten Datensatz in den Speicher zu laden. Deshalb wird ein Data Loader verwendet, welcher Stück für Stück die Daten dem Model zur Verfügung stellt. Dieser erwartet jedoch solch eine Datensatz-Klasse in einem vorher abgestimmten Format. Die Daten werden dort gespeichert und der Data Loader kann über die __getitem__ Methode der Klasse die Daten über einen Index laden.

Tokenize_headlines

Wie schon im Kapitel zu BERT erläutert benötigt BERT ein bestimmtes Format. Dazu gehören zum einen die Special Tokens, welche den Anfang und das Ende eines Satzes anzeigen. Außerdem müssen alle Sätze die gleiche Länge besitzen, weshalb nicht belegter Platz mit Padding aufgefüllt wird. Die Attention Mask zeigt dann an, an welchen Stellen tatsächlich Wörter sind und bei welchen es sich nur um Padding handelt. Die Methode tokenize_headlines bringt mit dem übergebenen Tokenizer den übergebenen Data Frame in dieses Format, erzeugt aus diesen Enkodierungen & Label ein Objekt der NewsSentimentDataset Klasse und gibt dieses dann zurück.

Datensatzaufbereitung

Auf die im letzten Abschnitt beschrieben Methode wird hier zurückgegriffen. Zuerst wird dazu der Tokenizer definiert, denn je nach ausgewähltem Modell ist ein anderer nötig. Da in der Projektarbeit das BERT-Modell „bert-base-cased“ genutzt wird, wird der entsprechende Tokenizer genutzt. Anschließend wird der gewählte Datensatz in einen Trainingsteil (80%) und einen Testteil (20%) aufgeteilt. Am Ende der Datensatzaufbereitung werden mit der Methode tokenize_headlines die Daten in das korrekte Format umgewandelt.

Training & Testing

```
model = BertForSequenceClassification.from_pretrained('bert-base-cased', num_labels=3)
model = model.to(device)
#data loader
train_batch_size = 8
val_batch_size = 8

train_data_loader = DataLoader(train_dataset, batch_size = train_batch_size, sampler=RandomSampler(train_dataset))
val_data_loader = DataLoader(val_dataset, batch_size = val_batch_size, sampler=SequentialSampler(val_dataset))

#optimizer and scheduler
num_epochs = 1
num_steps = len(train_data_loader) * num_epochs
optimizer = AdamW(model.parameters(), lr=5e-5, eps=1e-8)
#makes learning rate increase during warum up steps and decrease linearly during training.
scheduler = get_linear_schedule_with_warmup(optimizer, num_warmup_steps=0, num_training_steps=num_steps)

#training and evaluation
seed_val = 64
random.seed(seed_val)
np.random.seed(seed_val)
torch.manual_seed(seed_val)
torch.cuda.manual_seed_all(seed_val)
```

In dem Hauptteil des Programms findet die Definition des Models und der Parameter statt sowie das Training und Testing. Wie schon erwähnt wird hier das Model „bert-base-

cased“ genutzt. Es gibt auch ein größeres Modell mit mehr Parametern. Dieses bringt jedoch nicht viel mehr Performance bei weit größerem Rechenaufwand. In dem Forschungspapier von BERT wird für das Fine-Tuning eine Batchgröße von 16 bzw. 32, eine Learning Rate von 5e-5 bis 2e-5 und eine Epochenanzahl von 2 bis 4 mit dem Optimizer Adam empfohlen. In dieser Projektarbeit wird jedoch die Batchgröße von 8 genutzt auf Grund von Hardwarelimitierungen. Die beiden Data Loader werden definiert, wobei für das Training die Daten in zufälliger und für das Testing in sequenzieller Reihenfolge geladen werden. Anschließend werden weitere Parameter wie Epochenanzahl und Learning Rate definiert. Auch der Optimizer und Scheduler werden festgelegt. Ein Seed wird genutzt, um die Reproduzierbarkeit der Ergebnisse zu gewährleisten. Hierbei ist jedoch zu erwähnen, dass dies die Reproduzierbarkeit der Ergebnisse lediglich verbessert. Durch die zufällige Initialisierung der Parameter des Models am Anfang des Trainings weichen die Ergebnisse bei gleichen Hyperparametern dennoch leicht voneinander ab.

```
#training phase
average_train_loss = 0
average_train_acc = 0
for step, batch in enumerate(train_data_loader):

  model.train()
  input_ids = batch['input_ids'].to(device)
  attention_mask = batch['attention_mask'].to(device)
  labels = batch['labels'].to(device)

  # set the gradient to zero as by default pytorch is accumulating the gradients
  model.zero_grad()

  loss, logits = model(input_ids=input_ids,
                attention_mask=attention_mask,
                labels=labels)

  #loss is cross entropy loss by default
  average_train_loss += loss

  if step % 100 == 0:
    print("At Step {} Training Loss: {:.5f}".format(step, loss.item()))

  #backpropagation
  loss.backward()
  #maximum gradient clipping
  torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
  #update parameters
  optimizer.step()
  #update learning rate
  scheduler.step()

  logits_for_acc = logits.detach().cpu().numpy()
  label_for_acc = labels.to('cpu').numpy()
  average_train_acc += sklearn.metrics.accuracy_score(label_for_acc, np.argmax(logits_for_acc, axis=-1))

  #print out sentences + sentiment predictions + labels
  print(tokenizer.batch_decode(input_ids, skip_special_tokens=True))
  print("Predictions: ",np.argmax(logits_for_acc, axis=1))
  print("Labels:      ",label_for_acc)
  print("#############")


average_train_loss = average_train_loss / len(train_data_loader)
average_train_acc = average_train_acc / len(train_data_loader)
print("======Average Training Loss: {:.5f}=========".format(average_train_loss))
print("======Average Training Accuracy: {:.2f}%========".format(average_train_acc*100))
```

Daraufhin beginnt die Training- und Testphase. In jeder Epoche werden zuerst die Trainingsdaten durchlaufen. Das Modell wird in den Trainingsmodus versetzt. Da PyTorch die Gradienten automatisch akkumuliert müssen diese, um dies zu verhindern, auf null gesetzt werden. Die geladenen Daten werden in das Model gegeben und der Cross Entropy Loss wird (automatisch) berechnet. Anschließend wird Backpropagation betrieben und die Accuracy berechnet. Der Optimizer aktualisiert die Parameter und der Scheduler die Learning Rate.

Beim Testing werden mit Hilfe des Data Loaders die Testdaten durchgegangen ähnlich wie bei dem Training. Es werden jedoch keine Parameter verändert. Lediglich werden die Daten in das Model übergeben, um den Loss und die Accuracy zu berechnen und auszugeben. Hierdurch kann sichergestellt werden, dass das Model generalisiert und nicht in eine Überanpassung gerät. Im Folgenden ein Ausschnitt der Vorhersagen mit Labels sowie die Genauigkeiten mit dem Verlust. Da die dazugehörige Liste der Sätze, welche in der Batch klassifiziert wurden, in einer Zeile ausgegeben werden, sind sie hier abgeschnitten:

```
['Johnson & Johnson Completes Acquisition of Momenta Pharmaceuticals, Inc.', 'The Dows Changes Could Impact These Fund
Predictions:  [0 0 2 1 0 1 0 0]
Labels:       [0 2 2 2 2 1 0 0]
#############
['Why Kratos Defense Shares Are Falling Today', 'China to be added to FTSE global bond benchmark in 2021', 'Finnish ba
Predictions:  [1 0 2 0 1 2 0 2]
Labels:       [1 0 2 0 2 2 0 2]
#############
['Bahrain to Sell Dollar Bond, Sukuk to Plug Gaping Budget Deficit', 'Apple Gives Tim Cook Up to a Million Shares That
Predictions:  [1 0 0 1 0 1 1 0]
Labels:       [2 0 0 1 0 1 1 0]
#############
["` ` We hope to clarify our policies to Finnish businesses where there are any gray areas,'' Motlanthe said.", 'US ST
Predictions:  [2 0 0 2]
Labels:       [2 0 2 2]
#############
======Average Training Loss: 0.65881=========
======Average Training Accuracy: 71.83%========
======Average Validation Loss: 0.52171=========
======Average Validation Accuracy: 78.88%======
```

Nachdem dieser Teil abgeschlossen ist wird das Model als Datei abgespeichert. Welche Parameter bei welchem Datensatz zum besten Ergebnis führen wird im nächsten Kapitel erforscht.

Anmerkungen

Bei der Implementierung bin ich auf Schwierigkeiten beim Training gestoßen. Als ich die Daten in das Modell geladen und das Training gestartet hatte versuchte das Modell anfänglich von den Daten zu lernen. Dies war ihm jedoch nicht möglich. Das Resultat war, dass BERT alle Sätze als dieselbe Klasse klassifiziert. Verschiedene Dinge wurden versucht, um dies zu beheben: Änderungen der Hyperparameter, Verwendung verschiedener Datensätze, korrigieren von Fehlern im Code (die Labels wurden falsch geladen) und das Ausgleichen der

Verteilung der Klassen auf je 1/3 der Größe des Datensatzes. Nichts konnte jedoch das Problem lösen. Anschließend konnte jedoch das Programm selbst als Fehlerquelle ausgeschlossen werden: Die Transformers Bibliothek bietet ein „Trainer“ Modul an, welches eine vereinfachte Möglichkeit ist, ein Modell mit einem Datensatz zu trainieren. Dort trat das gleiche Problem wie bei meiner eigenen Implementierung auf, weshalb das Problem wohl aus einer anderen Quelle wie bspw. dem Datensatz kommen musste. Außerdem wurde eine Lösung mit „SimpleTransformers“, einer Bibliothek auf der Transformers Bibliothek aufbaut und ein stark vereinfachtes Interface zum Trainieren von Modellen bietet, versucht. Dies löste tatsächlich das Problem. Dadurch konnte ich die Anzahl an möglichen Fehlerquellen verringern. Nach einiger Zeit des Debuggings konnte ich den Fehler finden: Die Labels. Ich hatte die Labels folgendermaßen zugewiesen: negativ = 0, neutral = 1 und positiv = 2. Dies führte bei BERT jedoch, wie beschrieben, dass das Modell nicht richtig lernen konnte und möglicherweise in einem lokalen Minimum gefangen war. Die Änderung der Labels auf die momentane Aufteilung (positiv = 0, negativ = 1 und neutral = 2) konnte das Problem glücklicherweise lösen. Es ist jedoch für mich unverständlich warum das Problem auftrat. Möglich wäre, dass BERT die Annahme hat, dass positiv und negativ immer 1 und 0 sind (welche von den beiden Zahlen für welches Sentiment genutzt wird ist tatsächlich egal). Bei einer Klassifizierung von Nachrichten in bspw. Politik, Wirtschaft und Kultur lässt sich jedoch solch eine Bewertung nicht vornehmen, eine Nachrichtenkategorie ist nicht positiver als eine andere. Allgemein sollten die numerischen Bezeichnungen für die Klassen irrelevant für das Modell sein. Beim Training ist die Ausgabe vom Modell, abgesehen vom Loss (welcher nur automatisch berechnet wird, wenn zusätzlich die Labels in das Modell gegeben werden), ein n langes Array von Logits ausgegeben, wobei n die Anzahl der Klassen ist. Die Stelle im Array mit dem höchsten Wert ist die Vorhersage der Klasse des Modells. Das Modell entscheidet also mit einer gewissen Sicherheit, wie der Inputsatz zu klassifizieren ist. Dabei sollte die Reihenfolge der Klassen eigentlich keine Rolle spielen. Dies ist tatsächlich ein interessanter Fehler und könnte ein Bug in PyTorch oder der Transformers Bibliothek sein. Ich konnte jedoch weder einen Fehler in meinem Code (dieser funktioniert nun fehlerfrei) noch eine Erklärung zu dem Problem finden. Der Fehler bleibt also vorerst ohne Erklärung. Eine tiefgreifende Untersuchung des Problems wäre tatsächlich interessant und sinnvoll, jedoch auch sehr aufwändig, da Machine Learning Modelle recht schlecht nachvollziehbar agieren.

Kapitel 6: Resultate

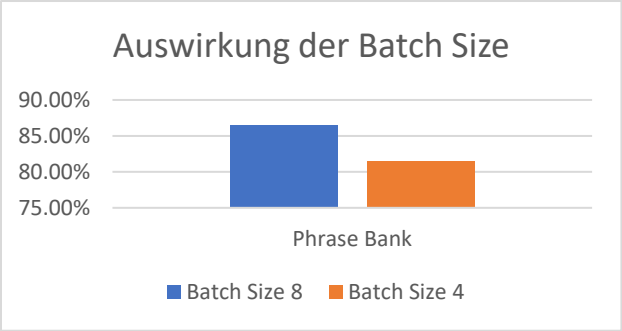
In diesem Kapitel werde ich die Resultate verschiedener Versuche präsentieren. Dabei soll gezeigt werden wie sich BERT bei verschiedenen Parametern und Datensätzen verhält. So kann dann ermittelt werden welches Modell die beste Performance bei der Sentiment Analysis von Wirtschaftsnachrichten erreicht. Hierbei wird zum einen als Optimizer stets Adam benutzt, da dieser zum Zeitpunkt des Schreibens den State-of-the-Art darstellt und auch in dem originalen Forschungspapier empfohlen wird. Als Batchgröße wird 8 benutzt, obwohl die Empfehlung bei 16 oder 32 liegt. Dies liegt an Hardwarelimitierungen. Eine noch kleinere Batchgröße führt zudem zu Performanceverlust, wie in den Experimenten erkennbar. Eine größere Batchgröße wie 12 wurde auch versucht und konnte nach dem Fine-Tuning tatsächlich für ein besseres Ergebnis sorgen. Jedoch begrenzt Google Colab, nach so einem rechenaufwändigen Training die zur Verfügung stehenden Ressourcen, weshalb es nur mit einer Batchgröße von 8 möglich war alle für die Experimente nötigen Trainingsdurchläufe zu unternehmen.

Die Versuche

Es wurden für jeden Datensatz die gleiche Reihe von unterschiedlichen Parametern ausprobiert, um zu sehen wie sich welche Parameter auf die Performance auswirken. Festgehalten wurden die gewählten Parameter sowie die Train- und Test Accuracy in der letzten Epoche. Zusätzlich wurde auch, wie schon erwähnt probiert eine geringere Batchgröße zu nehmen. Des Weiteren wurde auch ein Model des einen Datensatzes auf dem Datensatz eines anderen getestet, um zu sehen wie gut das Model generalisiert. Im Folgenden werden die Ergebnisse der verschiedenen Versuche dargestellt.

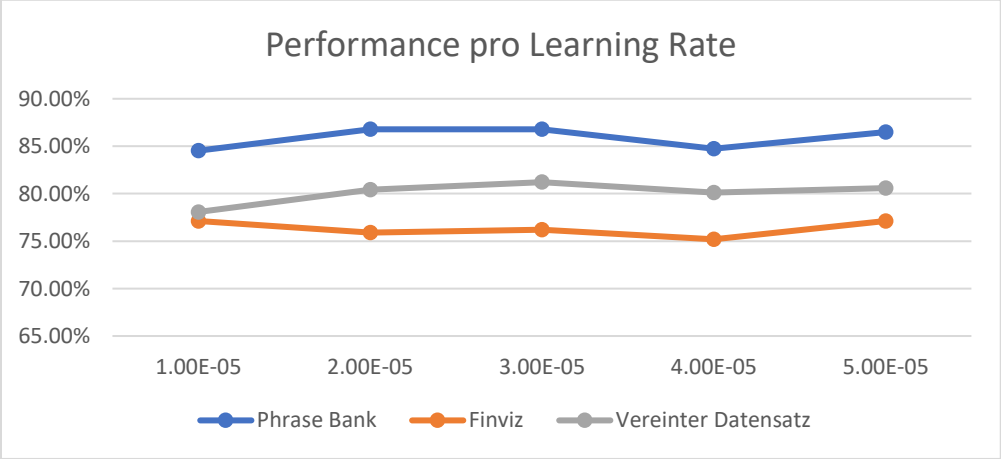
Batch Size

Dieser Versuch wurde lediglich unternommen, um beispielhaft die Performance zu vergleichen zwischen einer kleineren Batch Size von 4 und der in dieser Projektarbeit genutzten Batch Size von 8. Hier wird deutlich, dass die höhere Batch Size dem Model eine deutlich bessere Performance einbringt.



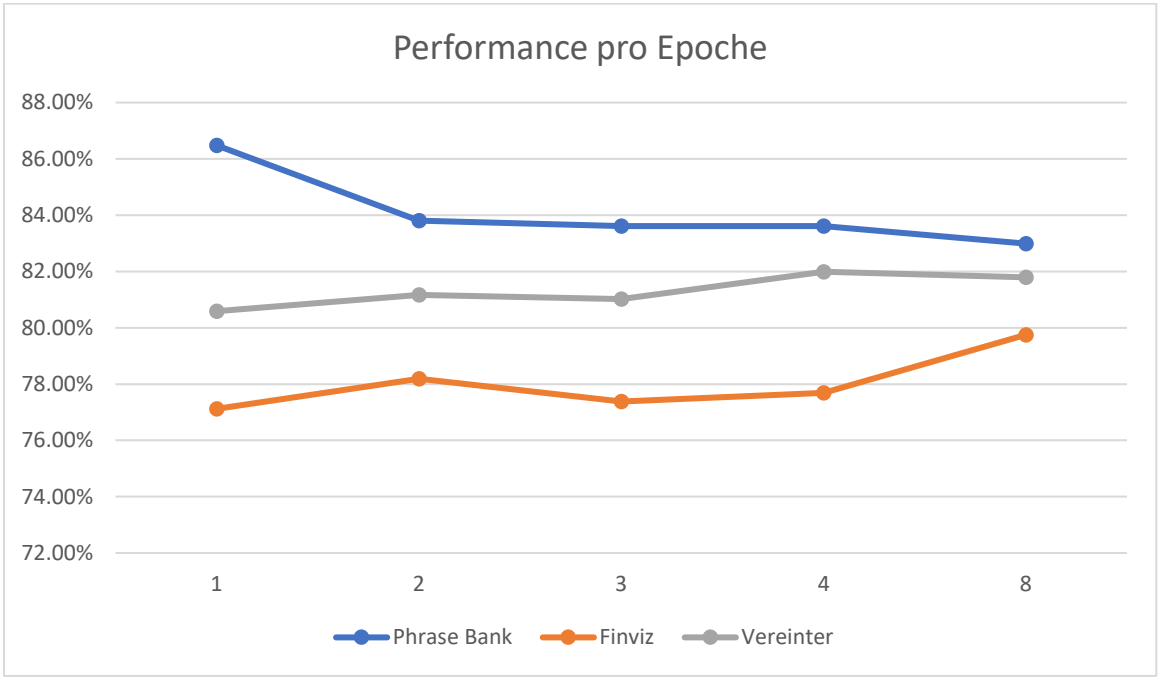
Learning Rate

Im folgenden Graphen werden die Ergebnisse der Experimente mit unterschiedlicher Learning Rate dargestellt. Das Training und Testing wird auf dem Phrase Bank Datensatz von Kaggle, meinen eigenen Datensatz, der die Quelle finviz.com nutzt, und dem vereinten Datensatz aus den beiden vorher genannten. Die angegebene Genauigkeit ist die Testgenauigkeit. Trainiert wurde immer eine Epoche ohne Warmup Steps und mit dem Optimizer Adam. Man sieht zum einen, dass das Modell, welches auf dem Phrase Bank Datensatz trainiert wurde auf diesem die beste Performance erzielt. Das Modell, das auf meinem Datensatz trainiert wurde weist eine 6-8% schlechtere Performance auf. Das vereinte Modell befindet sich im Bezug auf die Performance in der Mitte der beiden Modelle. Bei dem kleinsten Datensatz von allen, dem Phrase Bank Datensatz, wird bei einer Learning Rate von 2e-5 oder 3e-5 das beste Ergebnis erzielt. Bei dem vereinten Datensatz ist dies 3e-5. Hierbei ist besonders auffallend, dass die Performance weniger stark von einer Änderung der Learning Rate beeinflusst wird. Der Finviz Datensatz wird mit 1e-5 die beste Performance erzielt. Auch zu erwähnen ist, dass 5e-5 zwar nie die besten aber immer solide Ergebnisse erzielt. Eine Learning Rate von 5e-6 wurde auch versucht, jedoch ohne Performancegewinn. Die Daten dazu befinden sich im Appendix.



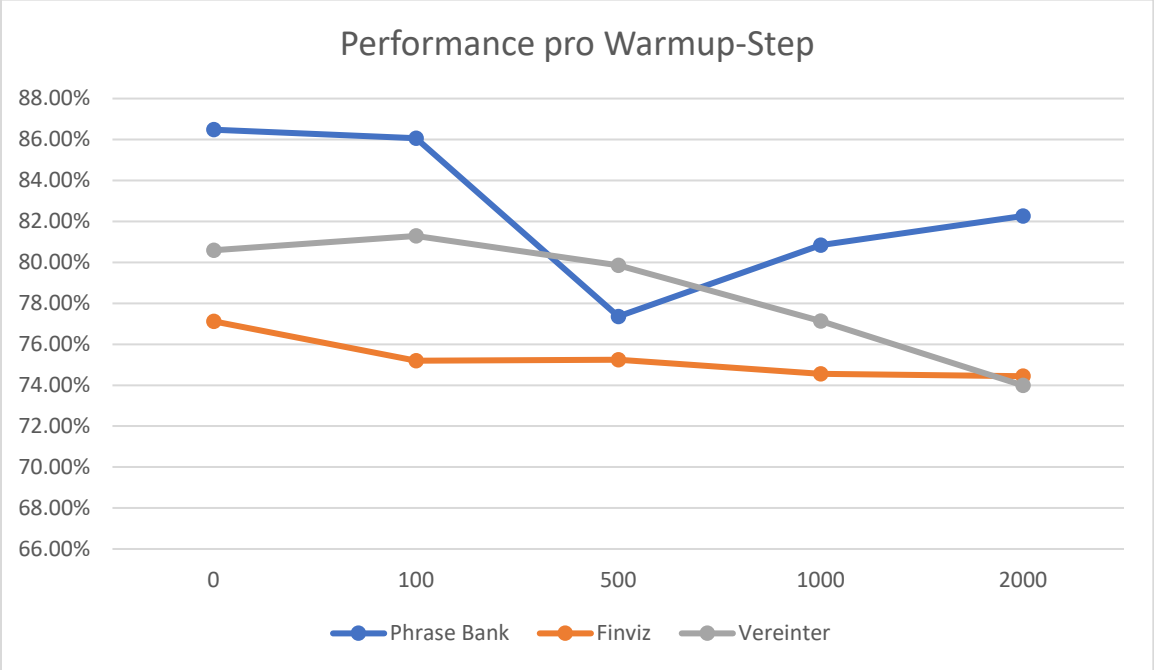
Epoche

Bei diesem Versuch soll erforscht werden, wie BERT bei unterschiedlichen Daten und verschieden Epochenanzahlen reagiert. Genutzt wurde eine Learning Rate von 5e-5, ohne Warmup Steps und mit dem Optimizer Adam. Trainiert und getestet wurde immer auf dem Datensatz für das jeweilige Model. Hierbei ist zu erkennen, dass eine höhere Epochenanzahl BERT bei einem kleinen Datensatz wie Phrase Bank eher schadet. Es ist zudem festzustellen, dass in diesem Fall auch die Trainingsgenauigkeit schneller ansteigt. Bei den anderen größeren Datensätzen steigt die Performance jeweils, wobei bei dem vereinten Datensatz BERT weniger empfindlich auf die Epochenanzahl reagiert.



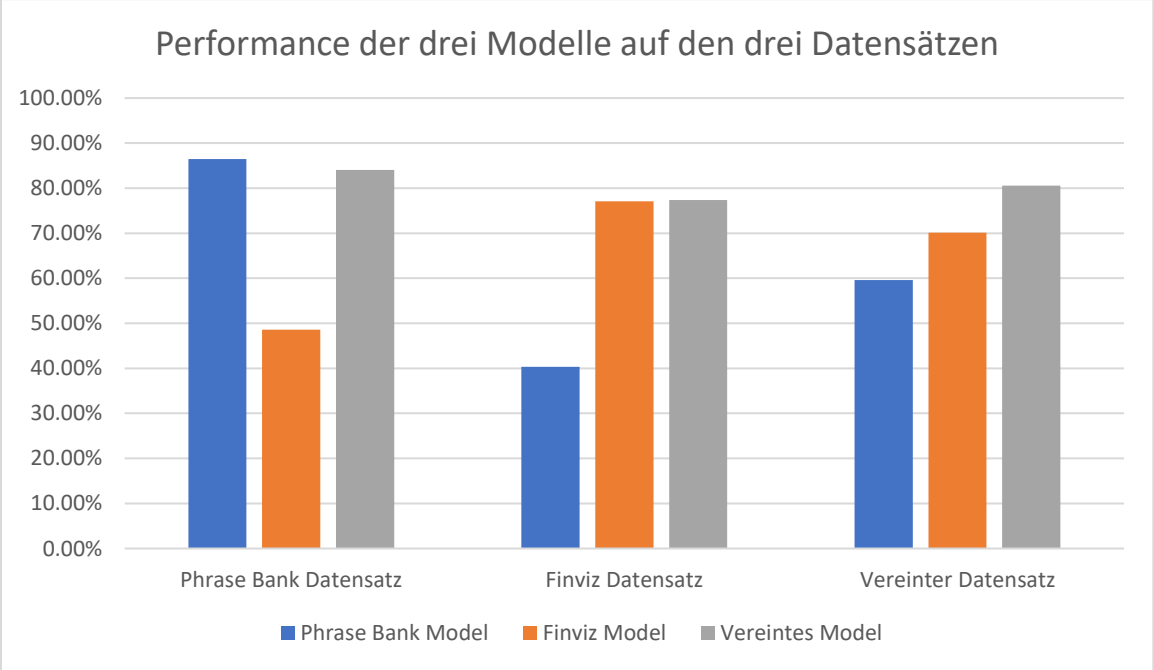
Warmup-Steps

Hier werden verschiedene Werte für den Parameter Warmup-Steps versucht, um zu erforschen, ob dieser Parameter genutzt werden kann, um die Performance zu verbessern. Trainiert wurde wieder eine Epoche lang mit eine Learning Rate von 5e-5 und dem Optimizer Adam. Hierbei ist klar erkennbar, dass es eine negative Korrelation zwischen der Performance und dem Paramter Warmup-Steps gibt. Beim Phrase Bank Datensatz schwankt die Performance, bei dem Finviz Datensatz bleibt sie recht stabil mit negativem Trend und bei dem vereinten Datensatz leidet die Performance am meisten, steigt jedoch zuerst bei einem Wert von 100.



Vergleich der Modelle

Dieser Versuch soll zum Vergleich der drei Modelle dienen. Hierbei wurde jeweils eine Epoche lang bei einer Learning Rate von 5e-5, 0 Warmup-Steps und dem Optimizer Adam das Modell trainiert. Jedes Modell wurde also auf jedem Testdatensatz der anderen Modelle getestet. Bei dem vereinten Datensatz wurde nur der Trainingsteil des zu testenden Datensatz mit dem anderen Datensatz vereint. So wird sichergestellt, dass die Testdaten komplett neu für das Modell sind. Hierbei sind mehrere Dinge festzustellen: Die Modelle weisen bei den Datensätzen, auf denen sie auch trainiert wurden, immer eine sehr gute Performance auf. Dabei ist anzumerken, dass das Modell vom vereintem Datensatz auf allen drei Datensätzen eine ähnlich gute Performance aufweist. Auf dem Phrase Bank Datensatz ist sie jedoch etwas höher als bei dem Finviz Datensatz. Auffallend sind außerdem die Performance der anderen beiden Datensätze, die jeweils nur auf einem der Datensätze trainiert wurde: Beide weisen eine deutlich schlechtere Performance beim Testen auf dem anderen Datensatz auf. Wobei das Model, das auf dem Finviz Datensatz trainiert wurde, sowohl auf dem vereintem Datensatz als auch auf dem Phrase Bank Datensatz eine bessere Performance aufweist als das Phrase Bank Model auf dem vereintem und Finviz Datensatz.



Schlussfolgerung

Aus den Beobachtungen in den Experimenten ziehe ich folgende Schlüsse: Die Wahl der Batch Size von 8 scheint zum Testen optimal zu sein bei der Abwägung zwischen Performance und Rechenaufwand. Wie schon erwähnt wurde auch eine Batchgröße von 12 versucht, wodurch ein Performanceanstieg von 80,59% auf 82,40% erzielt werden konnte. Obwohl auf Grund von Hardwarelimitierungen weitere Tests mit dieser Batchgröße nicht möglich waren zeigt dies, dass eine größere Batchgröße zu einer besseren Performance führt. Außerdem ist auffällig, dass bei größerem Datensatz das BERT Modell weniger anfällig für die Veränderung der Parameter ist. Zudem sollte bei großen Datensätzen eher eine hohe Epochenanzahl und bei kleinen eine niedrige gewählt werden, da je nach Datensatz die Training Accuracy schneller oder langsamer steigt. Somit kann bei einem kleinen Datensatz schon nach 2 Epochen Overfitting entstehen, wenn bei großen Datensätzen noch mehr Training benötigt wird. Auch ist ersichtlich, dass eine hohe Warmup-Step Anzahl eher eine niedrige Performance bringt, die Learning Rate je nach Datensatz jedoch mal mehr und mal weniger Performance bringt. Weiterhin ist ersichtlich, dass Modelle, die auf meinem Datensatz trainiert wurden, eine schlechtere Performance erzielen als Modelle beim Training und Testen auf dem Phrase Bank Datensatz. Dies hat folgende Gründe: Zum einen ist mein Datensatz nicht so sauber, wie der Phrase Bank Datensatz, da dieser von mehr Leuten mit mehr wirtschaftlicher Expertise erstellt wurde. Zum anderen weisen die Daten im Phrase Bank Datensatz weniger Diversität auf. Es sind viele ähnliche Sätze aufzufinden und der Finviz Datensatz ist von Natur aus diverser, da finviz.com seine Daten von verschiedenen Nachrichtenorganisationen erhält. Der Unterschied in den Datensätzen erklärt somit, warum die Modelle, die nur auf einem Datensatz trainiert wurden, beim Testen auf dem anderen Datensatz eine so schlechte Performance aufweisen. Das Fine-Tuning bei BERT scheint somit so fein zu sein, dass obwohl beide Datensätze Wirtschaftsnachrichten enthalten, die leicht andere Art der Daten das Modell verwirrt. Dies wird verdeutlicht bei der Betrachtung der Performance beim Testen auf dem vereintem Datensatz. Das Modell, das auf beiden Datensätzen trainiert wurde, weist bei allen Datensätzen eine solide Performance auf und scheint mir am meisten zu generalisieren, weshalb ich dieses Modell für die Sentiment Analysis wählen würde. Die beste Performance (82,40%) wurde hierbei mit folgenden Parametern erzielt: Batch Size = 12, Optimizer = Adam, Learning Rate = 5e-5, Epochenanzahl = 4, Warmup-Steps = 0. Hierbei gelten die Limitierungen, die im Abschnitt „Datensatz“ besprochen wurden. Ich halte jedoch, mit leichten Einschränkungen, das Modell für allgemein anwendbar zur Klassifizierung von Wirtschaftsnachrichten, da auf eine

Mischung aus allgemeinen und spezifischen Nachrichten, Nachrichten aus verschiedenen Sektoren sowie Quellen etc. geachtet wurde. Auch wenn ein Datensatz, welcher von mehreren Leuten annotiert wird, mehr einzelne Unternehmen und zeitlich sowie regional diverser ist wünschenswert wäre. Unterstützen möchte ich diese Schlussfolgerung durch einen Vergleich mit dem FinBERT Modell, ein auf BERT aufbauendes und auf dem Phrase Bank Datensatz trainierten Sprachmodell. Dabei wurde eine Genauigkeit von 86% erreicht, gleich mit der welche in dieser Projektarbeit erreicht wurde, obwohl bei FinBERT zusätzliche Daten zum Pre-Training genutzt wurden [42]. Meine Projektarbeit zeigt jedoch, dass der Phrase Bank Datensatz allein nicht genügt, um eine akkurate Sentiment-Klassifizierung für alltägliche Wirtschaftsnachrichten aus verschiedenen Quellen zu ermöglichen. Dies könnte an der Art der Daten, die Größe des Datensatzes, die Verteilung das Sentiment oder an einer Mischung aus verschiedenen Gründen liegen. Des Weiteren wird deutlich, dass durch eine Mischung des Phase Bank und Finviz Datensatzes ein akkurates, allgemeineres und solideres Modell erreicht werden kann.

Kapitel 7: Fazit

Zum Abschluss der Projektarbeit fasse ich kurz die Erkenntnisse dieser Projektarbeit zusammen. Es wurden die Rahmenbedingungen der Projektarbeit dargestellt sowie Grundlagen in der Sentiment Analysis und NLP. Besonders wurde auf die Fortschritte des Deep Learning im NLP Bereich eingegangen und auf BERT aufbauend eine Machine Learning Lösung entwickelt, die mit einer 82,40%igen Genauigkeit das Sentiment von Nachrichten aus verschiedenen Nachrichtenquellen im Finanzbereich vorhersagen kann. Besonders durch die Nutzung verschiedener Datenquellen konnte ein Model trainiert werden, welches zuverlässig verschiedene Nachrichtenüberschriften korrekt klassifizieren kann.

Ausblick

Bezüglich weiterer Entwicklungen lässt sich folgendes sagen: Um weitere Performanceverbesserungen zu erlangen, könnte zum einen mit besserer Hardware eine größere Batch Size genommen werden, wie im originalen Forschungspapier empfohlen. In der Projektarbeit wurde bereits gezeigt, dass dies zu deutlichem Performancegewinn führen kann. Auch mehr Parameterkombinationen könnten so ausprobiert werden. Weiterhin wäre ein Vergleich von verschiedenen transformerbasierten Modellen sinnvoll, die auf den Ideen von BERT aufbauen und weniger Rechenaufwand beim Training oder Unterstützung für Multilingualität bieten. Hierzu wäre ein Vergleich zwischen bspw. BERT, XLNet, RoBERTa und XLM interessant. Multilingualität würde es ermöglichen die Stimmung in verschiedenen Sprachen/Ländern zu messen und zu vergleichen. Schließlich würde die Erstellung eines separaten Testdatensatzes, welcher unbekannt für alle Modelle sind, für mehr Neutralität beim Vergleich der Modelle sorgen. Weitreichendere Analysen könnten auch sinnvoll sein. So könnte untersucht werden, ob manche Klassen besser erkannt werden als andere. Oder ob und wie stark eine Parteilichkeit in den Datensätzen und somit dem Modell vorhanden sind. Auch ein Vergleich zwischen Modellen die Groß- und Kleinbuchstaben unterscheiden mit Modellen, die dies nicht tun, könnte erkenntnisreich sein. Schlussendlich wäre es von Interesse einen neuen Datensatz zu entwickeln, der zeitlich sowie geographisch divers ist und eine Vielzahl von Nachrichten zur Wirtschaft und zu Einzelaktien aus verschiedenen Sektoren enthält, um qualitative Forschung zukünftig zu ermöglichen.

Appendix

Resultate der Experimente in Tabellenform:

Kaggle_Dataset:	Train_Batch	Val_Batch	Learning_Rate	Epoch	Warmup	Val_Acc	Train_Acc
	8	8	5,00E-05	1	0	86.48%	80.05%
	8	8	5,00E-05	2	0	83.81%	92.04%
	8	8	5,00E-05	3	0	83.61%	96.39%
	8	8	5,00E-05	4	0	83.61%	98.27%
	8	8	2,00E-05	1	0	86.78%	79.59%
	8	8	1,00E-05	1	0	84.53%	71.83%
	8	8	3,00E-05	1	0	86.78%	80.67%
	8	8	1,00E-03	1	0	59.53%	56.55%
	8	8	5,00E-05	1	100	86.07%	77.58%
	8	8	5,00E-05	1	500	77.36%	72.29%
	8	8	5,00E-05	1	1000	80.84%	67.14%
	8	8	5,00E-05	1	2000	82.27%	64.64%
	8	8	2,00E-05	6	1000	84.02%	99.18%
	8	8	5,00E-05	10	1000	83.30%	99.82%
	8	8	5,00E-05	8	0	82.99%	99.82%
	4	4	5,00E-05	1	0	81.48%	76.47%
Val on Merged	8	8	5,00E-05	1	0	59.63%	79.87%
	8	8	4,00E-05	1	0	84.73%	79.74%
Val on My Dataset	8	8	5,00E-05	1	0	40.31%	79.87%

My_Dataset:	Train_Batch	Val_Batch	Learning_Rate	Epoch	Warmup	Val_Acc	Train_Acc
	8	8	5,00E-05	1	0	77.12%	69.42%
	8	8	5,00E-05	2	0	78.19%	84.94%
	8	8	5,00E-05	3	0	77.38%	92.81%
	8	8	5,00E-05	4	0	77.69%	96.86%
	8	8	2,00E-05	1	0	75.88%	69.36%
	8	8	1,00E-05	1	0	73.56%	65.34%
	8	8	3,00E-05	1	0	76.19%	67.38%
	8	8	1,00E-03	1	0	43.44%	38.91%
	8	8	5,00E-05	1	100	75.19%	66.92%
	8	8	5,00E-05	1	500	75.25%	62.94%
	8	8	5,00E-05	1	1000	74.56%	61.14%
	8	8	5,00E-05	1	2000	74.44%	58.89%
	8	8	5,00E-05	6	1000	78.44%	99.12%
	8	8	5,00E-05	10	1000	78.44%	99.78%
	8	8	5,00E-05	8	0	79.75%	99.58%
Val on Merged	8	8	5,00E-05	1	0	70.15%	69.17%
	8	8	4,00E-05	1	0	75.19%	68.98%
Val on Kaggle	8	8	5,00E-05	1	0	48.57%	69.27%

Merged_Dataset:	Train_Batch	Val_Batch	Learning_Rate	Epoch	Warmup	Val_Acc	Train_Acc
	8	8	5,00E-05	1	0	80.59%	72.92%
	8	8	5,00E-05	2	0	81.17%	89.10%
	8	8	5,00E-05	3	0	81.02%	93.76%
	8	8	5,00E-05	4	0	81.99%	97.36%
	8	8	2,00E-05	1	0	80.40%	74.08%
	8	8	1,00E-05	1	0	78.07%	70.77%
	8	8	3,00E-05	1	0	81.21%	73.51%
	8	8	1,00E-03	1	0	36.65%	36.29%
	8	8	5,00E-05	1	100	81.29%	72.31%
	8	8	5,00E-05	1	500	79.85%	68.41%
	8	8	5,00E-05	1	1000	77.14%	65.29%
	8	8	5,00E-05	1	2000	73.99%	64.38%
	8	8	2,00E-05	6	1000	79.50%	98.83%
	8	8	5,00E-05	10	1000	80.98%	99.81%
	8	8	5,00E-05	8	0	81.79%	99.63%
Val on Kaggle	8	8	5,00E-05	1	0	84.02%	72.87%
	8	8	4,00E-05	1	0	80.10%	72.66%
Val on My Dataset	8	8	5,00E-05	1	0	77.38%	73.61%
Val on My Dataset	8	8	5,00E-05	4	0	79.12%	97.14%
Val on My Dataset	8	8	5,00E-06	4	0	77.50%	88.61%
Val on My Dataset	8	8	5,00E-06	8	0	79.31%	95.82%
Val on My Dataset	8	8	5,00E-06	12	0	79.19%	98.41%
	12	12	5,00E-05	4	0	82.40%	97.36%

Quellen

[1] „Wie Amazon lernte was wir wollten“ – Artikel über Algorithmen zur Produktempfehlung [Abruf: 16.03.20]

https://www.zeit.de/digital/2019-07/amazon-algorithmus-greg-linden-empfehlungssysteme

[2] “Political Campaigns Know Where You’ve Been. They’re Tracking Your Phone.” – Artikel über die Nutzung von Daten bei Wahlkampagnen [Abruf: 16.03.20]

https://www.wsj.com/articles/political-campaigns-track-cellphones-to-identify-and-target-individual-voters-11570718889

[3] „Algorithmischer Handel/HFT“ – Artikel zur Definition von algorithmischem Handel [Abruf: 15.03.20]

https://www.deutsche-boerse.com/dbg-de/regulierung/regulatory-dossiers/mifid-mifir/mifid-i-to-mifid-ii/market-structure/algo-hft

[4] „Algorithmic Trading Systems“ –Vorstellung von algorithmischen Handelssystemen [Abruf: 15.03.20]

https://seekingalpha.com/instablog/40574325-peter-knight/4926454-algorithmic-trading-systems

[5] Morton Glantz, Robert Kissell. *Multi-Asset Risk Modeling: Techniques for a Global Economy in an Electronic and Algorithmic Trading Era*. Academic Press, 2013

[6] Lin, Tom C. W., The New Investor. 60 UCLA Law Review 678 (2013); 60 UCLA Law Review 678 (2013); Temple University Legal Studies Research Paper No. 2013-45. https://ssrn.com/abstract=2227498

[7] “Trading Halts Don’t Stop Stock Market Carnage” – Aktien fallen stark nach Ankündigung von Trump [Abruf: 15.03.20]

https://www.nasdaq.com/articles/trading-halts-dont-stop-market-carnage-2020-03-12

[8] Bing Liu: Sentiment Analysis and Opinion Mining, Morgan & Claypool Publishers, 2012

https://www.cs.uic.edu/~liub/FBS/SentimentAnalysis-and-OpinionMining.pdf

[9] Bing Liu: Sentiment Analysis: Mining Opinions, Sentiments and Emotions, Cambridge University Press, 2015

[10] Eric Chu, Deb Roy: Audio-Visual Sentiment Analysis for Learning Emotional Arcs, 2017

https://ieeexplore.ieee.org/abstract/document/8215563

[11] “Innovations in Finance with Machine Learning, Big Data, and Artificial Intelligence” – Zusammenfassung der neusten Forschungen von J.P.Morgan [Abruf: 21.03.20]

https://www.jpmorgan.com/global/research/machine-learning

[12] Barbara Novick, Daniel Mayston, Sherry Marcus, Rachel Barry, Gassia Fox, Bradley Betts, Stefano Pasquali, Kyle Eisenmann: Artificial intelligence and machine learning in asset management, 2019

https://www.blackrock.com/corporate/literature/whitepaper/viewpoint-artificial-intelligence-machine-learning-asset-management-october-2019.pdf

[13] Min-Yuh Day, Chia-Chou Lee: Deep Learning for Financial Sentiment Analysis on Finance News Providers, 2016

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7752381

[14] “Introducing DeepText: Facebook’s text understanding engine” – Einführung in DeepText

[Abruf: 04.04.20]

https://engineering.fb.com/core-data/introducing-deeptext-facebook-s-text-understanding-engine/

[15] Nasukawa, Tetsuya, Jeonghee Yo: Sentiment Analysis: Capturing Favorability Using Natural Language Processing, 2003

[16] Das, Sanjiv and Mike Chen: Yahoo! For Amazon: Extracting Market Sentiment from Stock Message Boards, 2001

[17] Kamal Nigam, Andrew Kchites Mccallum, Sebastian Thrun, Tom Mitchell: Text Classification from Labeled and Unlabeled Documents using EM, 2000

[18] Marti A. Hearst: Direction-Based Text Interpretation as an Information Access Refinement, 1992

[19] Nhan Cach Dang, María N. Moreno-García, Fernando De la Prieta: Sentiment Analysis Based on Deep Learning: A Comparative Study, 2020

[20] Beschreibung des BoW und des TF-IDF Feature Modells [Abruf: 19.04.20]

https://medium.com/greyatom/an-introduction-to-bag-of-words-in-nlp-ac967d43b428

[21] Vorstellung des Stanford Log-Linear Part-Of-Speech Tagger [Abruf: 19.04.20]

https://nlp.stanford.edu/software/tagger.shtml

[22] Marzieh Saeidi, Guillaume Bouchard, Maria Liakata, Sebastian Riedel: SentiHood: Targeted Aspect Based Sentiment Analysis Dataset for Urban Neighbourhoods, 2016

[23] Fei Liu, Trevor Cohn, Timothy Baldwin: Recurrent Entity Networks with Delayed Memory Update for Targeted Aspecd-based Sentiment Analysis, 2018

[24] Yukun Ma, Haiyun Peng, Erik Cambria: Targeted Aspect-Based Sentiment Analysis via Embedding Commonsense Knowledge into an Attentive LSTM, 2018

[25] Rangliste von Sentiment Analysis Modellen zu bekannten öffentlichen Datensätzen [Abruf: 19.04.20]

http://nlpprogress.com/english/sentiment_analysis.html

[26] Offizielle Beschreibung des Word2Vec Moddels [Abruf: 25.04.20]
https://code.google.com/archive/p/word2vec/

[27] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean: Efficient Estimation of Word Representation in Vector Space, 2013

https://arxiv.org/pdf/1301.3781.pdf

[28] Zweite Vorlesung über RNNs aus Intro to Deep Learning vom MIT [Abruf: 26.04.20]

http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L2.pdf

[29] Sepp Hochreiter, Jürgen Schmidhuber: Long short-term memory, 1997

[30] Yoav Goldberg: Neural Network Methods in Natural Language Processing, 2017

[31] Erklärung von RNNs und LSTMs von Google Mitarbeitern [Abruf: 28.04.20]

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[32] Rafal Jozefowicz, Wojciech Zaremba, Ilya Sutskever: An Empirical Exploration of Recurrent Network Architectures, 2015
http://proceedings.mlr.press/v37/jozefowicz15.pdf

[33] Klaus Greff, Ripesh K. Srivastava, Jan Koutník, Bas R. Steunebrink, Jürgen Schmidhuber: LSTM: A Search Space Odyssey, 2015

https://arxiv.org/pdf/1503.04069.pdf

[34] Erklärung Exploding Gradient [Abruf: 27.07.20]

https://machinelearningmastery.com/exploding-gradients-in-neural-networks/

[35] Definition Gradient Clipping [Abruf: 27.07.20]

https://deepai.org/machine-learning-glossary-and-terms/gradient-clipping

[36] Performance von Aspect-Based Sentiment Analysis Ansätzen am Datensatz Sentihood [Abruf: 01.08.20]

https://paperswithcode.com/sota/aspect-based-sentiment-analysis-on-sentihood

[37] Jacob Devlin, Ming-Wie Chang, Kenton Lee, Kristina Toutanova: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, 2018

https://arxiv.org/pdf/1810.04805.pdf

[38] Beschreibung der Nachteile von RNNs und LSTMs [Abruf: 05.08.20]

https://towardsdatascience.com/the-fall-of-rnn-lstm-2d1594c74ce0

[39] Erklärung von Transformern [Abruf: 08.08.20]

https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04

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

[41] Detailliertere Erklärung von Transformern [Abruf: 08.08.20]

http://jalammar.github.io/illustrated-transformer/

[42] Yi Yang, Mark Christopher Siy UY, Allen Huang: FinBERT: A Pretrained Language Model for Financial Communications, 2020

[43] Vorstellung von BERT, ELMo und co [Abruf: 09.08.20]

http://jalammar.github.io/illustrated-bert/

[44] Pre-trained Language Models vorgestellt [Abruf: 09.08.20]

https://towardsdatascience.com/pre-trained-language-models-simplified-b8ec80c62217

[45] Toronto Deep Learning Series BERT Vorlesung [Abruf: 24.08.20]

https://dluo.me/TDLS_BERT

[46] Erklärung was XLNet besser macht als BERT [Abruf: 26.08.20]

https://medium.com/saarthi-ai/xlnet-the-permutation-language-model-b30f5b4e3c1e

[47] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, Ruslan Salakhutdinov: Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context, 2019

[48] Vorstellung und Erklärung von Transformer-XL auf Google AI Blog [Abruf: 26.08.20]

https://ai.googleblog.com/2019/01/transformer-xl-unleashing-potential-of.html

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

[50] Vorstellung von ELECTRA auf Google AI Blog [Abruf: 27.08.20]

https://ai.googleblog.com/2020/03/more-efficient-nlp-model-pre-training.html

[51] Erklärung des Durchbruchs in Machine Learning durch Self-Supervised Learning [Abruf: 31.08.20]

https://medium.com/intuitionmachine/the-paradigm-shift-of-self-supervised-learning-744a6819ce08

[52] Yann LeCun über die verschiedenen Lernmethoden von KI und ihre Zukunft [Abruf: 31.08.20]

https://bdtechtalks.com/2020/03/23/yann-lecun-self-supervised-learning/

[53] Fakten über Pre-Trained Language Models [Abruf: 20.10.20]

https://www.topbots.com/ai-nlp-research-pretrained-language-models/

[54] Vorstellung der 5 populärsten Programmiersprachen für Machine Learning [Abruf: 21.10.20]

https://www.geeksforgeeks.org/top-5-programming-languages-and-their-libraries-for-machine-learning-in-2020/

[55] Dokumentation der Bibliothek „requests“ [Abruf: 21.10.20]
https://pypi.org/project/requests/2.7.0/

[56] Dokumentation der Bibliothek „beautifulsoup“ [Abruf: 21.10.20]

https://www.crummy.com/software/BeautifulSoup/

[57] Dokumentation der Bibliothek „re“ [Abruf: 21.10.20]

https://docs.python.org/3/library/re.html

[58] Vorstellung der Bibliothek „pandas“ [Abruf: 22.10.20]

https://pandas.pydata.org/about/index.html

[59] Vergleich von Tensorflow und PyTorch [Abruf: 22.10.20]

https://medium.com/@UdacityINDIA/tensorflow-or-pytorch-the-force-is-strong-with-which-one-68226bb7dab4

[60] Vergleich der Popularität von Tensorflow und PyTorch [Abruf: 22.10.20]

https://towardsdatascience.com/is-pytorch-catching-tensorflow-ca88f9128304

[61] Dokumentation von der „transformers“-Bibliothek [Abruf: 22.10.20]

https://huggingface.co/transformers/

[62] Erklärung des Attention Layers [Abruf: 13.11.20]

https://d2l.ai/chapter_attention-mechanisms/transformer.html

[63] Kaggle Datensatz zu Wirschaftsnachrichten [Abruf: 14.11.20]

https://www.kaggle.com/ankurzing/sentiment-analysis-for-financial-news

[64] Malo, P., Sinha, A., Korhonen, P., Wallenius, J., & Takala, P.: Good debt or bad debt: Detecting semantic orientations in economic texts. Journal of the Association for Information Science and Technology, 2014

[65] Artikel über Performance & Kosten von Supervised ML sowie autonomen LKW [Abruf: 21.11.20]

https://medium.com/starsky-robotics-blog/the-end-of-starsky-robotics-acb8a6a8a5f5

[66] Narrow AI Definition [22.11.20]

https://deepai.org/machine-learning-glossary-and-terms/narrow-ai

[67] Bias in Machine Learning [28.11.20]

https://towardsdatascience.com/biases-in-machine-learning-61186da78591