Hochschule für Technik und Wirtschaft Dresden
Fakultät Informatik/ Mathematik

Abschlussarbeit zur Erlangung des akademischen Grades

Master of Science

Thema:

Adaption multilingual vortrainierter Modelle zur automatischen Zusammenfassung von Texten auf die deutsche Sprache

eingereicht von:	Daniel Vogel
eingereicht am:	13. August 2021
Erstgutachter:	Prof. Dr. Hans-Joachim Böhme
Zweitgutachter:	Dipl.-Kfm. Torsten Rex

Autorenreferat

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet.

Abstract

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ADAM	Adaptive Momentum Estimation
ATS	Automatic Text Summarization
BART	Denoising Sequence-to-Sequence Pre-training for NLG
BERT	Bidirectional Encoder Representations from Transformers
BLEU	Bilingual Evaluation Understudy
BOW	Bag-of-Words
BPE	Byte-Pair-Encoding
CUDA	Compute Unified Device Architecture
DL	Deep Learning
ELMo	Embeddings from Language Models
GBERT	German BERT
GloVe	Global Vectors for Word Representation
GPT	Generative Pre-Trained Transformer
GRU	Gated Recurrent Units
LCS	Longest Common Subsequence
LR	Learning Rate
LSTM	Long-Short-Term-Memory-Networks
ML	Machine Learning
MLM	Masked Language Models
MLP	Multi-Layer-Perceptron
NLG	Natural Language Generation
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
NLU	Natural Language Understanding
OHE	One-Hot-Encoding
REFZ	Referenzzusammenfassung
RL	Reinforcement Learning
RNN	Recurrent Neural Networks

ROUGE Recall-Oriented Understudy for Gisting Evaluation

SOTA State-of-the-Art

SYSZ Systemzusammenfassung

TL Transfer Learning

W2V Word2Vec

XLM-R Cross-Lingual Language Model RoBERTa

Quellcodeverzeichnis

1 Einleitung

Die Automatic Text Summarization (ATS) ist dem Bereich des Natural Language Processing (NLP) zuzuordnen und gewinnt zunehmend an wissenschaftlicher Relevanz. Obgleich entsprechende Modelle mittlerweile nicht mehr völlig neuartig sind, weisen die Entwicklungen der vergangenen Jahre qualitativ noch viele Potenziale auf [Yang et al., 2019, S. 1-2]. Einsatzmöglichkeiten entsprechender ATS-Modelle sind beispielsweise die Zusammenfassung von Nachrichten, die Zusammenfassung von Gesprächsprotokollen oder auch die Generierung von Überschriften, um nur wenige zu nennen [Goncalves, 2020]. Ziel ist in jedem Fall die Verdichtung von Informationen und die Reduktion der Lesezeit, wie Abbildung 1.1 demonstriert.

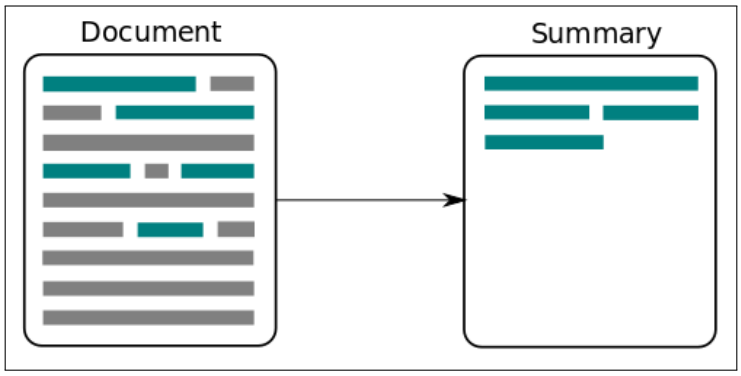

Abbildung 1.1: Ablauf einer automatischen Zusammenfassung [Thaker, 2019].

Mit besonderem Fokus auf das Gesundheitswesen lassen sich weiterhin zwei konkrete Einsatzgebiete konstruieren, in denen ein ATS-Modell in einem ganzheitlichen System als autarkes Modul implementiert werden könnte. Einerseits ist die Zusammenfassung von Patientengesprächen denkbar, wenn eine entsprechende Spracherkennung mit integrierter Sprechererkennung vorgeschaltet ist. Die verdichteten Informationen ließen sich anschließend zum Beispiel in Patientenakten exportieren oder anderweitig klassifizieren. Andererseits können Pflegeroboter, welche mitunter demente Patienten betreuen, durch ein ATS-Modell mit notwendigem Kontextwissen für die anstehenden Gespräche ausgestattet werden.

Die Anforderungen an ein ATS-Modell lassen sich aus dem individuell anvisierten Einsatzgebiet ableiten und können anhand verschiedener Faktoren klassifiziert werden. Demnach kann man prinzipiell zwischen dem extraktiven und dem abstraktiven Ansatz differenzieren [Gambhir et al., 2016, S. 5]. Extraktive Methoden bewerten die Sätze des ursprünglichen Textes anhand wort- und satzbezogener Attribute. Die Zusammenfassung entsteht sodann aus dem bewertungsgerechten Kopieren dieser Sätze [Kiani, 2017, S. 205-207]. Abstraktive Methoden hingegen verwenden Deep-Learning-Algorithmen, um Informationen zu identifizieren und entsprechende Zusammenfassungen mit völlig neuen Sätzen zu generieren [Nitsche, 2019, S. 1]. Weiterhin ist zu entscheiden, ob einzelne oder mehrere Dokumente zusammengefasst werden sollen, welcher Domäne diese Dokumente entstammen und ob möglicherweise eine Dialogorientierung vorliegt.

Aus technischer Sicht kommen bei der ATS grundsätzlich Sequence-to-Sequence-Modelle zum Einsatz. Dabei wird stets eine Eingabesequenz $x = [x_1, ..., x_n]$ in eine Ausgabesequenz $y = [y_1, ..., y_m]$ überführt, wobei n die Eingabelänge und m die Ausgabelänge ist. Die Sequenzen werden von Vektoren repräsentiert. Mithin wird bei der ATS $m < n$ intendiert. Sequenzen bestehen hierbei aus Symbolen, also etwa Zeichen, Zeichenketten oder auch Ziffern. Architekturen modellieren also die bedingte Wahrscheinlichkeit $P(y \mid x)$ [Nitsche, 2019, S. 32-33]. Die maßgebliche Herausforderung ist hierbei zum einen, dass ATS-Modelle tatsächlich die wichtigsten Informationen einer Eingabesequenz identifizieren. Zum anderen gilt es, diese Informationen in eine entsprechende Ausgabesequenz zu integrieren. Eben diese Ausgabesequenz ist zudem orthographisch und grammatikalisch korrekt zu generieren. Üblicherweise wird dieser Vorgang auch als Paraphrasierung bezeichnet. Menschen müssen diese Fähigkeit ebenfalls erst einmal erlernen.

1.1 Zielsetzung

Das Ziel dieser Arbeit ist dementsprechend die abstraktive Zusammenfassung einzelner Dokumente, wobei multilingual vortrainierte Modelle mittels Transfer Learning (TL) auf die deutsche Sprache adaptiert werden. Die Arbeit ist somit außerdem eine potenzielle Grundlage für die beiden konstruierten Einsatzgebiete aus dem Gesundheitswesen. Die Adaption auf die Domäne oder auch die Dialogorientierung ist nicht Teil dieser Arbeit.

Die Forschungsfragen lauten wie folgt:

- Wie lassen sich Texte automatisiert zusammenfassen?
- Wie können bereits existierende Modelle auf eine andere Sprache adaptiert werden?
- Wie qualitativ und skalierbar ist die Lösung?

1.2 Aufbau der Arbeit

Nach der Einleitung werden zunächst die Grundlagen des Deep Learning (DL) und des NLP offengelegt. Im Kapitel des DL werden neuronale Netze als solches definiert und ausgewählte Architekturen, welche auf die Zielerreichung einwirken, vorgestellt. Die Eigenschaften und die Relevanz von Hyperparametern und von TL schließen sich an. Im Kapitel des NLP werden neben der prinzipiellen Arbeit mit natürlicher Sprache und der entsprechenden Vorverarbeitung insbesondere sogenannte Word Embeddings und Deep Language Representations thematisiert. Bevor die bis dahin behandelten Komponenten in ein tatsächliches Modell integriert werden können, ist die Beschreibung des Forschungsstandes und der Datengrundlage erforderlich. Daran anschließend werden verschiedene Experimente durchgeführt und evaluiert. Der entsprechende Quellcode wird in Python entwickelt. Abbildung 1.2 stellt den Aufbau der Arbeit dar. Hier werden gleichzeitig die kapitelübergreifenden Zusammenhänge deutlich.

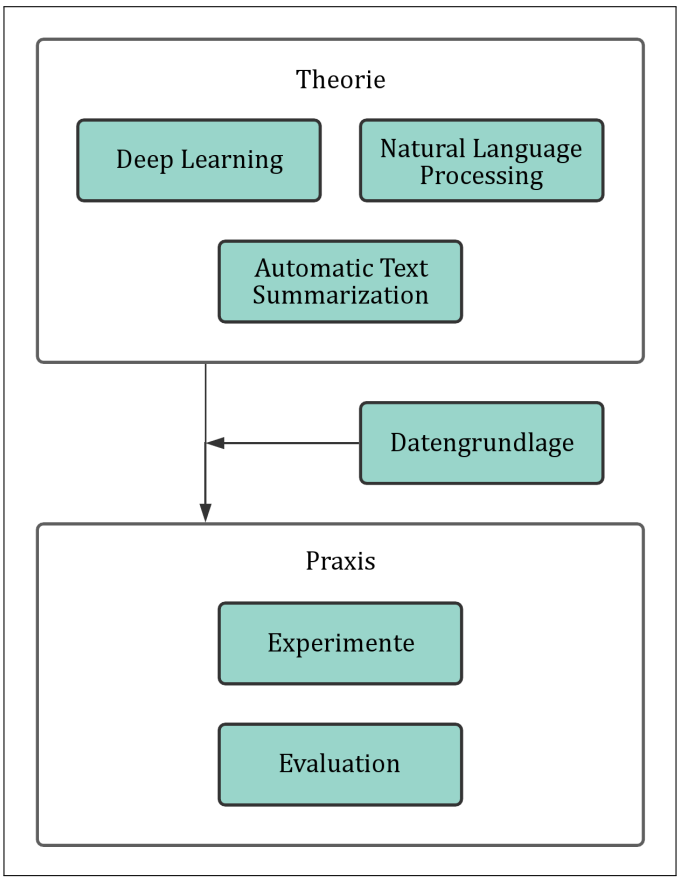

Abbildung 1.2: Aufbau der Arbeit.

1.3 Forschungsstand & Referenzen

Aufgrund der stetig fortschreitenden Entwicklungen überholt sich der Forschungsstand der ATS regelmäßig. Dennoch haben sich in den vergangenen Jahren gewisse Tendenzen erkennen lassen. Bereits zur Jahrtausendwende existierten erste ATS-Systeme. Waren die ersten Ansätze zumeist noch extraktiv, wurde sich in den vergangenen Jahren mehr und mehr auf die abstraktiven Ansätze konzentriert. Vor 2016 schienen Ansätze mit Recurrent Neural Networks (RNN) und Long-Short-Term-Memory-Networks (LSTM) sehr populär [Nallapati et al., 2016]. In den Jahren 2016 und 2017 etablierten sich Ansätze, welche auf Reinforcement Learning (RL) basierten [Paulus et al., 2017]. Seit 2018 legten diverse Ansätze mit Encoder-Decoder-Architekturen die Grundlage des heutigen State-of-the-Art (SOTA) [Yang et al., 2019, Rothe et al., 2020], denn um den SOTA konkurrieren fast ausschließlich sogenannte Transformer. Diese basieren auf den Encoder-Decoder-Architekturen, implementieren verschiedenartige Attention-Mechanismen und

haben sich sowohl unter qualitativen als auch unter ökonomischen und ökologischen Aspekten bewiesen [Zhang et al., 2020]. Diese Arbeit wird daher ebenfalls diesen Ansatz verfolgen, darüber hinaus jedoch die bislang unzulänglich behandelte Adaption auf die deutsche Sprache behandeln.

Die Qualität der ATS kann mithilfe des sogenannten ROUGE-Scores evaluiert werden. Dieser wird ebenso wie andere noch unerklärte Architekturen in einem nachfolgenden Kapitel dieser Arbeit umfangreich erläutert und kann zunächst als gegeben betrachtet werden. Die folgenden ROUGE-Scores können als zu reproduzierende Vergleichswerte verstanden werden: R-Recall: 15.96, R-Precision: 10.34, R-Measure: 12.22.

Weiterhin hat der Durchbruch frei verfügbarer vortrainierter Modelle die NLP-Welt revolutioniert, wie beispielsweise Bidirectional Encoder Representations from Transformers (BERT) [Devlin et al., 2019] sowie diverse Weiterentwicklungen. Verschiedenste NLP-Aufgaben wie die ATS konnten hiervon sehr stark profitieren. Die konkreten Funktionsweisen werden ebenfalls im Verlauf dieser Arbeit offengelegt. Wissenschaftliche Publikationen, welche mit dieser Arbeit vergleichbar sind und in dieser Arbeit oftmals referenziert werden, lauten wie folgt:

- Text Summarization with Pre-Trained Encoders [Yang et al., 2019]
- German Abstractive Text Summarization using Deep Learning [Nitsche, 2019]
- Leveraging Pre-Trained Checkpoints for Sequence Generation [Rothe et al., 2020]

Eben genannte Informationen werden in einem entsprechenden Kapitel zur ATS im Verlauf der Arbeit nochmal vertieft, wenn entsprechende Grundlagen kenntlich gemacht wurden. Dort werden der Forschungsstand und vergleichbaren Arbeiten separat thematisiert sowie entsprechende Verbesserungen hervorgehoben.

2 Deep Learning

Deep Learning ist ein Teilbereich des Machine Learning (ML). ML-Algorithmen analysieren Daten automatisiert mittels mathematischer Methoden der Mustererkennung. DL-Algorithmen bedienen sich hingegen vielschichtiger und hoch parametrisierter neuronaler Netze, um dem menschlichen Gehirn bestmöglich nachzuempfinden [Khanna, 2019, S. 455-457]. Dabei werden sehr große Datenmengen verarbeitet und analysiert, um einen Lerneffekt zu erzielen. Neben einer Eingabe- und einer Ausgabeschicht sorgen insbesondere die verborgenen Schichten für die prädizierte Tiefe. Hier werden Informationen weiterverarbeitet, abstrahiert und reduziert [Zhang et al., 2020, S. 131]. Die potenziellen Einsatzmöglichkeiten gehen über die der ML-Algorithmen hinaus. Der Aufbau neuronaler Netze sowie deren Funktionsweise und ausgewählte Architekturen werden in diesen Kapitel thematisiert. Hyperparameter und TL schließen sich an.

2.1 Neuronale Netze

Um den Aufbau und die Funktionsweise neuronaler Netze verstehen zu können, bedarf es zunächst der Beschreibung von Neuronen. Diese können im biologischen Sinne als Schalter verstanden werden, welche verschiedene Signale empfangen können und aktiviert werden, sobald genug Signale registriert wurden. Diese Aktivierung sendet folglich weitere Signale an andere Neuronen, wie Abbildung 2.1 im technischen Sinne exemplarisch skizziert [Kriesel, 2005, S. 42]. Hierfür werden Aktivierungsfunktionen benötigt, welche die gewichteten Eingangssignale in ein Ausgangssignal konvertieren. Sie ermöglichen es, nicht-lineare Zusammenhänge zwischen den Eingangs- und den Ausgangsdaten herzustellen [Zhang et al., 2020, S. 134].

Die elementarste Form neuronaler Netze wird Multi-Layer-Perceptron (MLP) genannt. MLP bestehen aus mehreren Schichten, deren Neuronen jeweils vollständig mit den Neuronen der umliegenden Schichten verbunden sind [Zhang et al., 2020, S. 131]. Der Verständlichkeit halber veranschaulicht Abbildung 2.2 einen solchen Aufbau mit nur einer verborgenen Schicht (engl. Hidden Layer), welche aus fünf Neuronen besteht. Dabei

zeichnen sich vollvermaschte Schichten (engl. Fully Connected Layer oder Dense Layer) dadurch aus, dass alle Neuronen mit allen Inputs und Outputs verbunden.

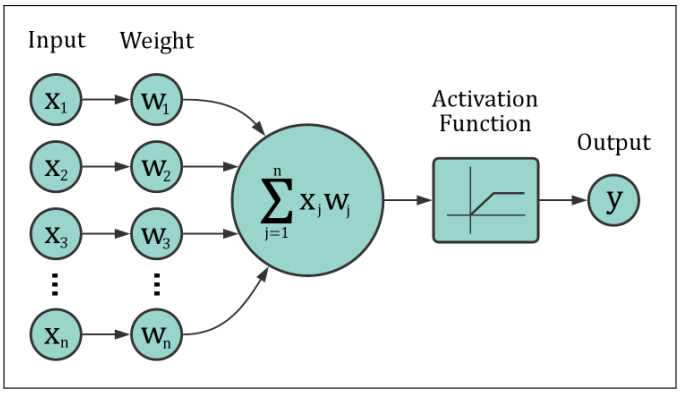

Abbildung 2.1: Aufbau eines künstlichen Neurons [McCullum, 2020].

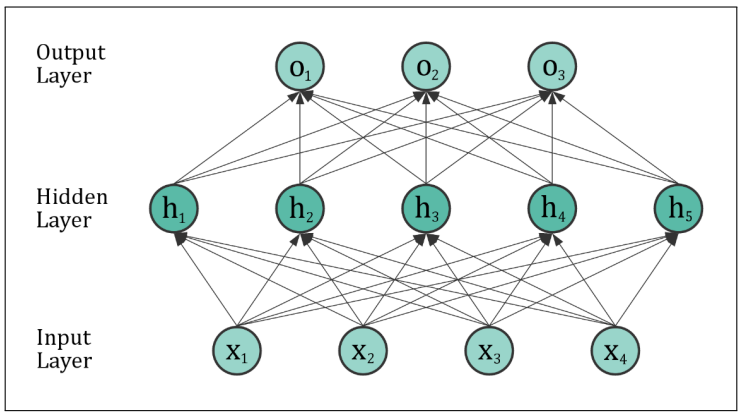

Abbildung 2.2: Aufbau eines MLP [Raschka et al., 2019, S. 388].

Ziel der hoch parametrisierten neuronalen Netze ist es, komplexe Funktionen hohen Grades bestmöglich zu approximieren und so verschiedenste Probleme zu lösen. Der anvisierte Lerneffekt wird mithilfe des sogenannten Backpropagation-Algorithmus erreicht. Hierbei werden Eingangsdaten zunächst vorwärts durch ein neuronales Netz hindurch propagiert. Mithilfe einer Fehlerfunktion wird sodann die erwartete mit der tatsächlichen Ausgabe verglichen und bewertet. Demzufolge sind gelabelte Daten erforderlich, um das hier beschriebene überwachte Training (engl. Supervised Learning) ausführen zu können [Raschka et al., 2019, S. 3]. Über das Gradientenverfahren werden die Fehler nun rückwärts durch das neuronale Netz propagiert und somit die Gewichte in den Neuronen angepasst, insbesondere in den verborgenen Schichten. Ziel ist die Minimierung der Fehlerfunktion und letztlich die Optimierung der durch das neuronale Netz

approximierten Funktion. Abbildung 2.3 verdeutlicht nochmals die Funktionsweise von überwachten Lernalgorithmen, welche in Folge eines Trainingsprozesses unbekannte Daten verarbeiten können [Zhang et al., 2020, S. 140, 169].

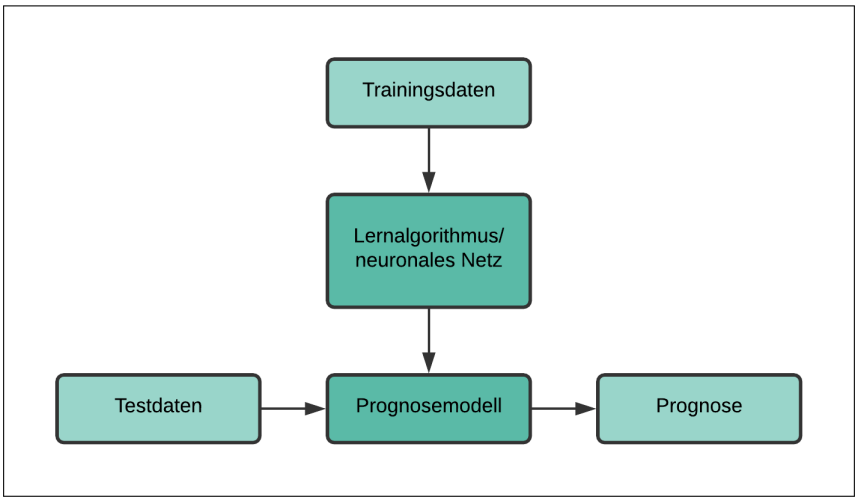

Abbildung 2.3: Supervised Learning [Raschka et al., 2019, S. 3].

Der Trainingsprozess erfolgt optimalerweise über mehrere sogenannte Epochen. Hier werden dem neuronalen Netz verschiedene Eingangsdaten zugeführt und beidseitige Propagationen ausgeführt. Wichtig ist dennoch, kein Overfitting beziehungsweise Underfitting zu erzeugen. Dies würde bedeuten, dass das trainierte Modell zu sehr oder zu wenig auf die Trainingsdaten angepasst ist. Ziel ist ein möglichst hoher Generalisierungseffekt des Modells, wie Abbildung 2.4 zeigt. Das Modell sollte den Lernfortschritt auf unbekannte Daten adaptieren können und darauf eine hohe Genauigkeit erreichen. Es gibt verschiedene Ansätze, um beispielsweise Overfitting vorzubeugen. Hier seien insbesondere Batch Normalization, Dropout und Early Stopping genannt, wobei entsprechende Mechanismen an anderweitiger Stelle erläutert werden [Zhang et al., 2020, S. 143-149].

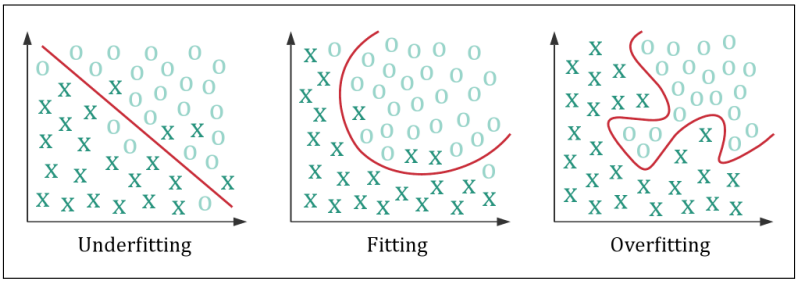

Abbildung 2.4: Typen von Generalisierungseffekten [Edpresso, O. J.].

2.2 Architekturen

Um mithilfe neuronaler Netze die ATS zu modellieren, werden nun ausgewählte Architekturen vorgestellt. Diese gehen weit über die als Grundlage beschriebenen MLP hinaus und verdeutlichen die Varietät neuronaler Netze.

2.2.1 Recurrent Neural Networks

Eingangsdaten der ATS haben in jedem Fall einen Textcharakter. Hierbei ist die Reihenfolge der Wörter hinsichtlich eines ausreichenden Textverständnisses von großer Bedeutung. Dies gilt sowohl für die Texte selbst als auch für die darin enthaltenen Sätze. Daher werden nun RNN vorgestellt, welche derart sequenzielle Daten mithilfe von zahlreichen verborgenen Zuständen in verborgenen Schichten verarbeiten können [Zhang et al., 2020, S. 301].

Sei h_t ein solcher verborgener Zustand und x_t ein Wort der Eingabesequenz an einem Index $t > 0$. Der Index ist hierbei als Position eines Wortes innerhalb eines Textes zu verstehen [Vaswani et al., 2017]. RNN können hierfür die Wahrscheinlichkeit

$$P(x_t \mid x_{t-1}, ..., x_1) \approx P(x_t \mid h_{t-1})$$

modellieren. Der verborgene Zustand h_t kann unter Kenntnis des Wortes x_t und des verborgenen Zustandes h_{t-1} gemäß der daraus herleitbaren Funktion

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

berechnet werden. Letzterer verfügt über die sequenziellen Informationen bis zum Index $t-1$. Weiterhin ist erkennbar, dass die gegebene Funktion der Markoveigenschaft gerecht wird [Zhang et al., 2020, S. 323-324].

Abbildung 2.5 demonstriert den rechentechnischen Vorgang anhand von drei prototypischen verborgenen Zuständen unter gegebenen Eingangsdaten. Wie bereits bei der oben genannten Funktion ersichtlich wurde, gehen in einen verborgenen Zustand jeweils das momentane Wort und der vorhergegangene verborgene Zustand ein. Innerhalb eines verborgenen Zustandes werden diese beiden Eingangsgrößen mithilfe einer Aktivierungsfunktion in eine vollvermaschte Schicht überführt. Zudem wird die Ausgabeschicht mit dem momentanen Output versorgt [Zhang et al., 2020, S. 325]. Allgemein haben RNN also die Form einer Kette, in der Module neuronaler Netze immer wiederkehren [AI-United, 2019].

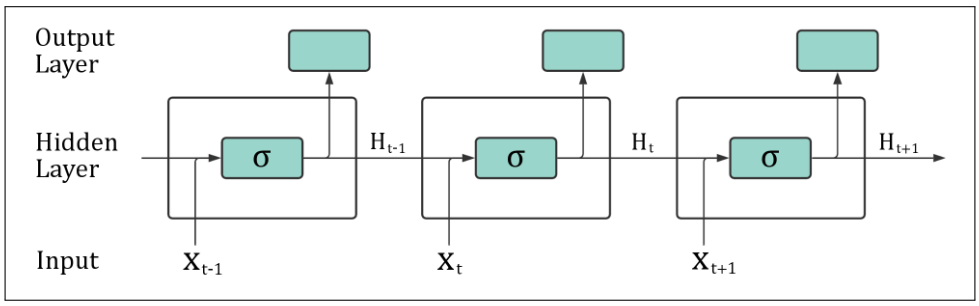

Abbildung 2.5: RNN mit verborgenen Zuständen [Zhang et al., 2020, S. 325].

Die bereits bekannte beidseitige Propagation ist auch bei RNN erforderlich, damit der Trainingsprozess zu einem entsprechenden Lernfortschritt führt. Während die vorwärtige Propagation den oben beschriebenen Prinzipien nachkommt, wird die Backpropagation nun über die Indizes hinweg durchgeführt (engl. Backpropagation Through Time). Bei einer langen Eingabesequenz gehen entsprechend hochdimensionale Matrizen in die Berechnung ein. Dies ist sowohl aus rechentechnischer als auch aus numerischer Sicht nachteilig. Die Anzahl modellinterner Parameter steigt bei steigender Indexanzahl hingegen nicht [Zhang et al., 2020, S. 328, 340].

Außerdem können RNN bidirektional modelliert werden. Die verborgenen Zustände werden demnach stets mit den Werten vor und nach dem momentanen Index berechnet. Diese beidseitige Propagation ist allerdings sehr rechenaufwendig und wird in dieser Arbeit nicht weiter vertieft.

RNN treffen früher oder später auf das Problem verschwindender Gradienten (engl. Vanishing Gradients). Dies tritt insbesondere bei langen Eingabesequenzen auf und führt dazu, dass langfristige Abhängigkeiten nicht mehr gelernt werden können. Die modellinternen Parameter werden im Trainingsprozess bekanntermaßen mithilfe der Backpropagation aktualisiert. Aufgrund fortschreitender Multiplikationen mit Wahrscheinlichkeiten zwischen 0 und 1 werden die Gradienten immer kleiner, bis letztlich kein Lernfortschritt mehr zu verzeichnen ist [Arbel, 2018]. Dieses Verhalten kann auch als numerischer Unterlauf bezeichnet werden.

LSTM sind eine Art von RNN, welche dazu fähig sind, langfristige Abhängigkeit zu erlernen. Sie vermeiden das Problem verschwindender Gradienten [AI-United, 2019]. Außerdem verfolgen sie das Prinzip sogenannter Gated Recurrent Units (GRU). Der entscheidende Mechanismus lässt aus der Funktionsweise von Gates ableiten. Demnach

entscheiden diese Gates mit einer Wahrscheinlichkeit zwischen 0 und 1, ob der erhaltene Wert weitergeleitet oder verworfen werden soll. Eine LSTM-Zelle verfügt über drei wesentliche Gates: Input Gate, Forget Gate, Output Gate [Zhang et al., 2020, S. 347-348].

Das Input Gate definiert, welche Werte in der Zelle berücksichtigt werden sollen. Hierzu gehören gemäß RNN das momentane Wort sowie der vorhergegangene verborgene Zustand. Das Forget Gate trifft eine Entscheidung darüber, ob und welche Werte gespeichert oder vergessen werden sollen. Hierfür eignen sich beispielsweise Aktivierungsfunktionen [Zhang et al., 2020, S. 355]. Über die intern festgelegte Logik kann somit der Zellzustand berechnet werden, unter anderem mithilfe punktweiser Matrixoperationen. Der Zellzustand wird sodann als neuer verborgener Zustand im Output Gate emittiert [Luber et al., 2018]. Der Aufbau einer LSTM-Zelle ist in Abbildung 2.6 zu sehen.

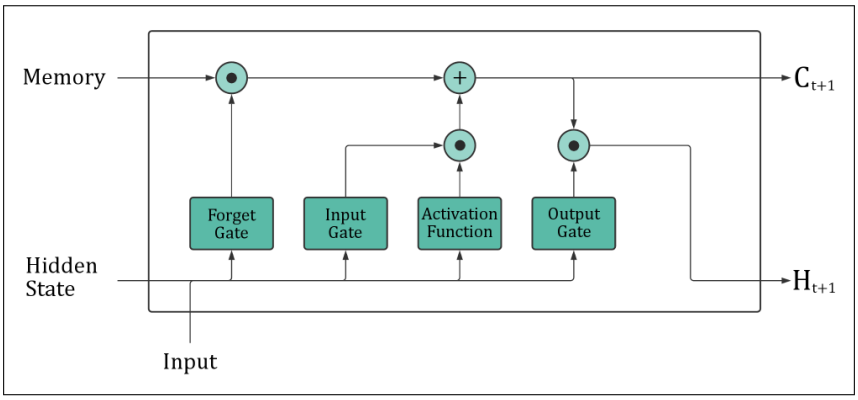

Abbildung 2.6: Aufbau einer LSTM-Zelle [Zhang et al., 2020, S. 357].

Ersetzt man die verborgenen Zustände in Abbildung 2.5 durch LSTM-Zellen, so wird man der beabsichtigten Funktionsweise eines LSTM gerecht. Mit fortschreitendem Trainingsprozess werden auch hier sequenzielle Abhängigkeiten gelernt, wobei als irrelevant angenommene Informationen stets verworfen werden. Der Vorteil gegenüber schlichter RNN besteht bei den LSTM hierdurch insbesondere darin, dass langfristige Abhängigkeiten besonders gut gelernt und bestimmte Informationen schneller gefunden werden können. Dies gilt vor allem für vielschichtige komplexe neuronale Netze, welche anderenfalls unübersichtlich und rechenaufwändig würden [Luber et al., 2018].

Zwar sind LSTM den RNN qualitativ in vielen NLP-Aufgaben überlegen, dennoch sind beide Architekturen eher statistisch getrieben und somit autark als veraltet zu bewerten. Obgleich sie als Grundlage prinzipiell verstanden werden sollten, dominieren maschi-

nelle Lernverfahren qualitativ sowie ressourcentechnisch den SOTA mit verschiedenen anderweitigen Architekturen [Culurciello, 2018]. Diese werden nachfolgend hinreichend erläutert.

2.2.2 Encoder-Decoder-Networks

Encoder-Decoder-Architekturen sind zunächst einmal als Template zu verstehen, welches einer stets individuellen Entwicklung und Reife bedarf. Dabei bestehen entsprechende Modelle aus einem Encoder und einem Decoder [Zhang et al., 2020, S. 375-376]. Beide Module bestehen de facto aus neuronalen Netzen, welche beispielsweise durch RNN oder auch LSTM repräsentiert werden können. Hierdurch wird zugleich die Verarbeitung sequenzieller Daten ermöglicht. Hinsichtlich der anvisierten ATS spricht man daher auch von Sequence-to-Sequence-Modellen [Zhang et al., 2020, S. 377].

Im Encoder wird die Eingabesequenz zuerst eingebettet. Hierdurch entsteht ein Merkmalsvektor, welcher entlang eines zugrundeliegenden Wortschatzes aus den Indizes der eingegangenen Wörter besteht. Er ist die mathematisch verarbeitbare Version der Eingabesequenz. Dieser Vorgang wird im weiteren Verlauf dieser Arbeit noch hinreichend beschrieben und untersucht. Der Merkmalsvektor geht sodann in das neuronale Netz des Encoders ein und wird in eine entsprechende Zustandsrepräsentation überführt [Yang et al., 2019].

Der Decoder wird mit den Ausgangsdaten des Encoders initialisiert. Die entsprechende Zustandsrepräsentation wird ebenfalls mithilfe eines neuronalen Netzes verarbeitet [Zhang et al., 2020, S.379]. Nun wird jedoch zusätzlich eine Ausgabesequenz generiert, welche der ATS gerecht werden soll. Wie bereits bekannt ist, gilt es letztlich die bedingte Wahrscheinlichkeit $P(y \mid x)$ zu modellieren [Yang et al., 2019].

Es folgt nun eine mathematische Betrachtung der Encoder-Decoder-Architektur. Hierfür wird die genannte Zustandsrepräsentation als Kontextvektor c bezeichnet. Die Wahrscheinlichkeit für eine Ausgabe am Index $t > 0$ kann demnach mit

$$P(y_t \mid y_1, ..., y_{t-1}, c)$$

modelliert werden. Die Berechnung der verborgenen Zustände im Decoder erfordert nun den Merkmalsvektor, den Kontextvektor und den letzten verborgenen Zustand des Encoders. Hiermit kann

$$h_t = f(y_{t-1}, c, s_{t-1})$$

berechnet werden. Informationen, welche an vorherigen Indizes gespeichert sind, können rekursiv ermittelt werden. Architektonisch ist weiterhin zu beachten, dass die Konfiguration des Encoders der Konfiguration des Decoders gleicht. Dies gilt insbesondere für die Anzahl der verborgenen Schichten [Zhang et al., 2020, S. 379].

Um die theoretisch und abstrakt beschriebene Architektur zu veranschaulichen, werden die wesentlichen Module nun abschließend in Abbildung 2.7 visuell in Zusammenhang gebracht. Allgemein gilt: Eine Eingabesequenz $x = [x_1, ..., x_n]$ wird mithilfe des Encoders zunächst in einen kontinuierlichen Zustandsvektor $z = [z_1, ..., z_n]$ überführt, bevor der Decoder daraus die Ausgabesequenz $y = [y_1, ..., y_m]$ generieren kann [Vaswani et al., 2017].

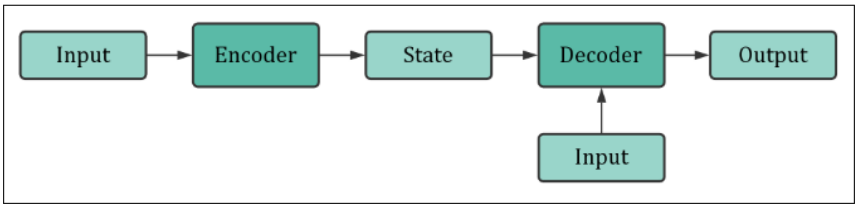

Abbildung 2.7: Encoder-Decoder-Architektur [Zhang et al., 2020, S. 375].

2.2.3 Attention in Neural Networks

Die Encoder-Decoder-Architektur kann im Kontext der ATS um einen sogenannten Attention-Mechanismus erweitert werden, welcher den Encoder mit dem Decoder verbindet [Vaswani et al., 2017]. Dabei geht es, wie die Übersetzung schon verrät, um Aufmerksamkeit. In der kognitiven Neurowissenschaft wird Aufmerksamkeit als ein Zustand gesteigerter Anspannung definiert, welcher selektive Wahrnehmung sowie entsprechendes Denken und Handeln umfasst. Diese Fähigkeit wird von einem ATS-Modell verlangt und mithilfe des Attention-Mechanismus realisiert. Um letztlich eine qualitative Zusammenfassung generieren zu können, selektiert der Attention-Mechanismus die wichtigsten Informationen aus dem Encoder, indem er die dort verarbeitete Eingabesequenz stets beobachtet und globale Zusammenhänge zwischen der Eingabesequenz und der Ausgabesequenz herstellt. Der Decoder wird dementsprechend darüber informiert. Das ATS-Modell soll mathematisch also menschenähnlichem Verhalten nachempfinden, weshalb es per Definition als KI-basierter Ansatz bezeichnet wird [Zhang et al., 2020, S. 389].

Der Attention-Mechanismus basiert indes auf einer Attention-Funktion, welche eine Query und ein Menge von Key-Value-Paaren einem Output zuordnet. Der Output wird dabei vektorweise als gewichtete Summe der Werte berechnet, während die Query determiniert, welchen Werten mehr Attention zugeordnet wird [Vaswani et al., 2017]. **??** zeigt eine solche Attention-Schicht.

Die Funktionsweise des Attention-Mechanismus innerhalb einer Encoder-Decoder-Architektur kann nun mithilfe Abbildung 2.8 weiter vertieft werden, wobei hier Index t betrachtet wird. Dabei bettet der Encoder die Eingabesequenz in bekannter Weise ein und verarbeitet sie, indem die verborgenen Zustände über alle verborgenen Schichten hinweg berechnet werden. Die Attention-Schicht erhält in der Folge alle Informationen, die der Encoder verarbeitet hat. Der Decoder wird nicht nur über den vorangegangenen verborgenen Zustand des Encoders informiert, sondern auch über den aus der Attention-Schicht resultierenden Kontext. Dieser wird als Antwort auf eine Query generiert, wobei eben diese Query wiederum durch den vorangegangenen verborgenen Zustand des Decoders repräsentiert wird [Zhang et al., 2020, S. 394]. Die Ausgabesequenz wird hierbei indexweise und autoregressiv generiert, da dem Decoder in jedem Index zusätzlich die bereits generierten Wörter zugeführt werden [Vaswani et al., 2017].

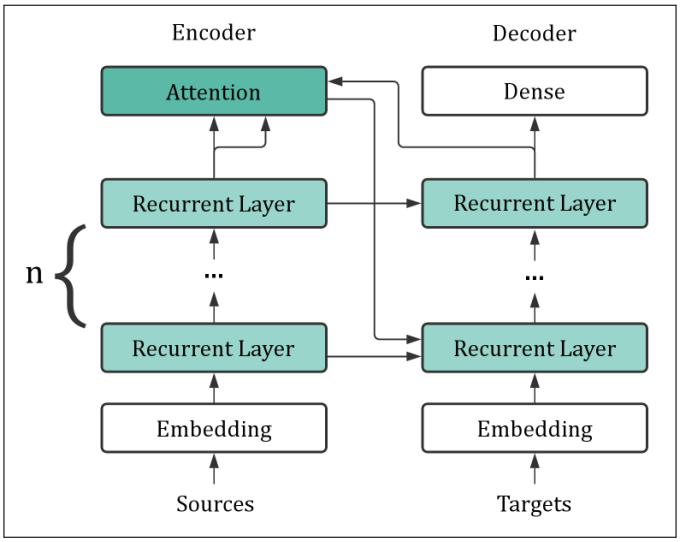

Abbildung 2.8: Attention-Mechanismus [Zhang et al., 2020, S. 394].

Weiterhin wird zwischen zwei Eigenarten unterschieden: Self-Attention und Multi-Head-Attention. Self-Attention transformiert innerhalb einer Query n Inputs in n Outputs. Dabei interagieren alle Inputs miteinander, um die Verteilung der globalen Attention zu

bestimmen. Die Outputs entstehen folglich, indem die entsprechenden Scores aggregiert werden. Betrachtet man also einen Satz, dann ist die Attention jedes darin enthaltenen Wortes zu berechnen [Karim, 2019]. Abbildung 2.9 visualisiert die Self-Attention architektonisch.

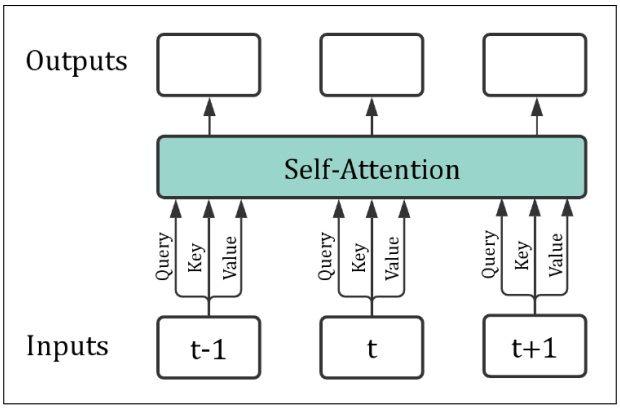

Abbildung 2.9: Self-Attention [Zhang et al., 2020, S. 400].

Multi-Head-Attention hingegen betrachtet direkt mehrere Queries. Die Matrizen der Queries Q, Keys K und Values V werden mithilfe entsprechender Gewichtsmatrizen dimensional reduziert, um

$$Head = Attention(QW^Q, KW^K, VW^V)$$

zu berechnen. Diese Berechnung geschieht h-mal, sodass entsprechend viele Heads in Form von Gewichtsmatrizen entstehen. Diese werden konkateniert und wiederum mit entsprechenden Gewichtsmatrizen transformiert, sodass

$$MultiHead(Q, K, V) = Concat(Head_1, ..., Head_h) \cdot W^O$$

gilt. Hierdurch wird es dem Modell ermöglicht, Informationen aus verschiedenen Repräsentationen zu identifizieren. Betrachtet man also erneut einen Satz, dann werden die Informationen positionsunabhängig und satzübergreifend identifiziert [Vaswani et al., 2017]. Abbildung 2.10 visualisiert die Multi-Head-Attention architektonisch.

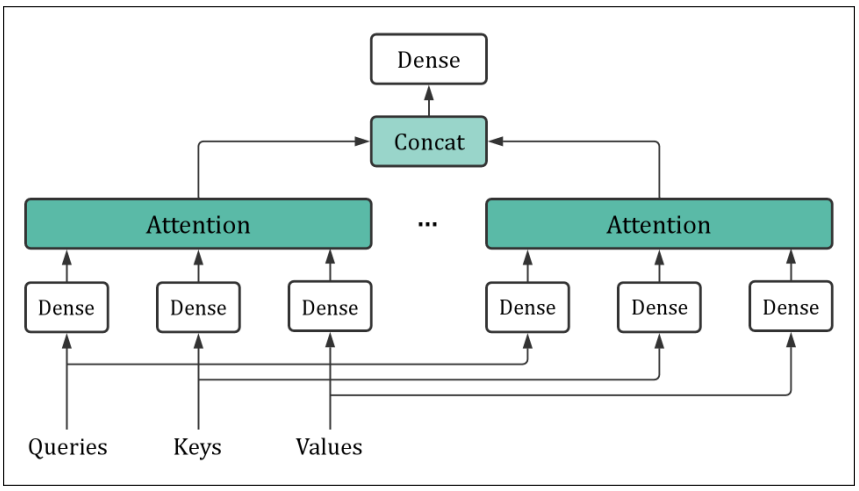

Abbildung 2.10: Multi-Head-Attention [Zhang et al., 2020, S. 400].

2.2.4 Transformer Networks

Transformer basieren ebenfalls auf der Encoder-Decoder-Architektur, wobei darüber hinaus verschiedene Attention-Mechanismen implementiert werden. Besonders ist hierbei, dass die Eingabesequenz parallel zur Anwendung der Attention-Mechanismen positionsabhängig eingebettet wird, um sequenzielle Informationen zu extrahieren. Dies führt insgesamt zu einem recht kompatiblen Modell, welches nur noch einer stark verringerten Trainingszeit bedarf [Zhang et al., 2020, S. 398].

Architektonisch werden die rekurrenten Schichten bisheriger Sequence-to-Sequence-Modelle durch Transformer-Module ersetzt. Diese bestehen aus einer Multi-Head-Attention-Schicht, einem positionsabhängigen Feed-Forward-Netzwerk und einer Layer-Normalization-Schicht. Eben diese wird benötigt, um für das entsprechende Modell einen generalisierenden Effekt zu erzielen. Transformer-Module analysieren die eingehenden Wörter unabhängig voneinander. Daher ist es wichtig, die Eingabesequenz positionsabhängig einzubetten, wie oben bereits angedeutet wurde. Hierdurch können sequenzielle Informationen extrahiert werden. Dabei werden überdies keine neuen Abhängigkeiten erlernt, wohl aber die Trainingszeit weiter reduziert [Zhang et al., 2020, S. 399-404]. Abbildung 2.11 visualisiert die Architektur im Vergleich zu Abbildung 2.8.

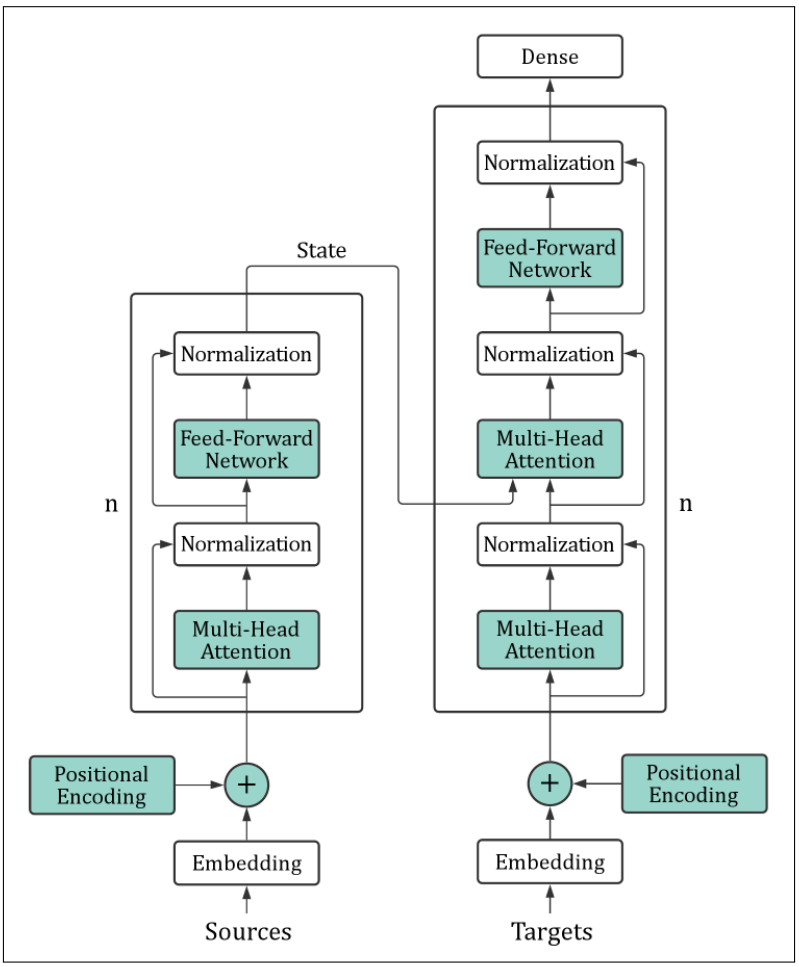

Abbildung 2.11: Transformer-Architektur [Zhang et al., 2020, S. 399].

Zuletzt ist wichtig, dass Transformer und ihre Komponenten verschiedenartig konzipiert werden können. Dies wird stets durch das anvisierte Ziel bedingt. Eine hinsichtlich der ATS geeignete Architektur wird in einem entsprechenden Kapitel noch umfangreicher offengelegt. Zudem können bestimmte Komponenten der Transformer durch vortrainierte Modelle repräsentiert werden. Dies wird ebenfalls im weiteren Verlauf dieser Arbeit thematisiert, nachdem entsprechende Grundlagen dargelegt wurden.

2.3 Hyperparameter

Hyperparameter sind Parameter einer Architektur, die bereits vor dem eigentlichen Trainingsprozess definiert werden. Sie bedürfen einer separaten Optimierung, da sie eben dieses Training und folglich auch die Qualität des entstehenden Modells enorm beeinflussen. Ziel ist es hierbei, die beste Kombination aller Hyperparameter zu finden, um die Fehlerfunktion hinreichend zu minimieren [Yang et al., 2020, S. 1].

Dies wird im Trainingsprozess als Teil der Backpropagation durch das Gradientenverfahren erreicht, welches die methodische Lösung allgemeiner Optimierungsprobleme übernimmt. Entlang eines negativen Gradienten wird das globale Minimum der dazugehörigen Fehlerfunktion gesucht, bis keine numerische Verbesserung mehr zu verzeichnen ist [Zhang et al., 2020, S. 428]. Im weiteren Verlauf werden ausgewählte Hyperparameter, welche das Gradientenverfahren und damit den allgemeinen Trainingsprozess hochgradig beeinflussen, mehr oder minder tiefgründig vorgestellt.

Die Learning Rate (LR) ist ein Hyperparameter, der bestimmt, wie viel Einfluss jede einzelne Epoche im Trainingsprozess auf die Anpassung der Gewichte nimmt. Sie gilt mithin als wichtigster Hyperparameter einer Architektur [Zhang et al., 2020, S. 428]. Eine zu niedrige LR kann den Trainingsprozess entweder stark verlangsamen oder dafür sorgen, dass kein Lernfortschritt mehr erzielt wird, da lokale Minima der Fehlerfunktion nicht übersprungen werden können und fälschlicherweise als globales Minimum interpretiert werden. Eine zu hohe LR kann hingegen sehr abrupte Anpassungen der Gewichte verursachen, sodass potenziell auch das globale Minimum übersprungen werden kann [Zhang et al., 2020, S. 414-415]. Abbildung 2.12 verdeutlicht diese Bedingungen. Ziel ist allgemein eine möglichst schnelle Konvergenz.

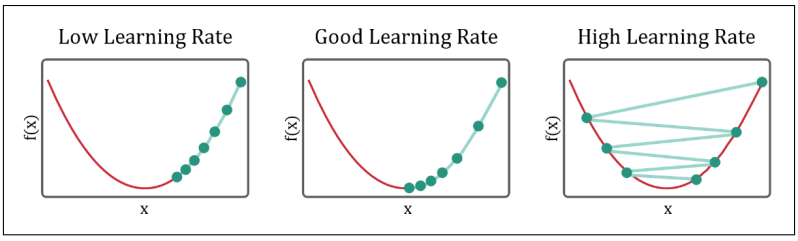

Abbildung 2.12: Konvergenzverhalten im Gradientenverfahren [Zhang et al., 2020, S. 429].

Neben der sorgfältigen manuellen Auswahl der LR, etwa mithilfe eines sogenannten LR-Schedule, ist es weiterhin möglich, eine adaptive LR einzuführen. Hierbei wird die LR in jeder Epoche verändert. Üblich ist hier eine Reduktion der LR, wenn bereits akzeptable Ergebnisse erreicht wurden [Zhang et al., 2020, S. 433].

Außerdem existiert das stochastische Gradientenverfahren, welches pro Epoche nur eine Stichprobe der verfügbaren Trainingsdaten berücksichtigt und einen generalisierenden Effekt verspricht [Zhang et al., 2020, S. 437]. Die Größe der Stichprobe wird üblicherweise als Batch Size bezeichnet und an dieser Stelle nur als weitergehender Hyperparameter genannt [Zhang et al., 2020, S. 446].

Weiterhin unterstützt das Momentum die bereits beschriebene LR auf der Suche nach dem globalen Minimum in der Fehlerfunktion. Dabei berücksichtigt es den Durchschnitt vorheriger Gradienten. Auf dieser Grundlage wird entschieden, in welche Richtung das stochastische Gradientenverfahren weiter absteigen soll, wie Abbildung 2.13 zeigt. Das Momentum ist somit potenziell in der Lage, lokale Minima zu überspringen und die Suche erst im tatsächlichen globalen Minimum zu beenden [Zhang et al., 2020, S. 453-456].

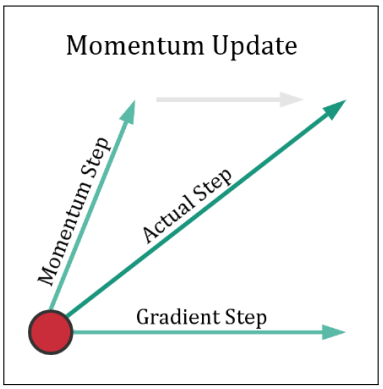

Abbildung 2.13: Gradientenverfahren unter Einfluss eines Momentums [CS231N, O. J.].

Bei der Auswahl eines hohen Momentums sollte die LR eher niedriger sein, oder anders herum. Eine Möglichkeit der stochastischen Optimierung ist hierbei Adaptive Momentum Estimation (ADAM). Dieser Algorithmus übernimmt nicht nur die Auswahl der adaptiven LR, sondern auch die Auswahl des entsprechenden Momentums. ADAM arbeitet weitreichenden Analysen zufolge effizient für daten- und parameterintensive Probleme. Dabei konvergiert der Algorithmus üblicherweise schneller als vergleichbare Optimierungsalgorithmen [Kingma et al., 2017, S. 1-2].

Zuletzt ist noch das Weight Decay erwähnenswert. Dieses meint die Multiplikation der Gewichte einer Architektur nach jeder Epoche mit einem Faktor kleiner als eins, um sehr große Gewichte zu verhindern. Die Gefahr von Overfitting wird hierbei verringert, während sich die Generalisierung des Modells verbessert [Zhang et al., 2020, S. 154]. Allgemein lässt sich die optimale Kombination aller Hyperparameter auch durch Techniken wie Grid Search (vgl. Brute-Force) annähern [Yang et al., 2020, S. 24].

2.4 Transfer Learning

TL ist in den letzten Jahren wissenschaftlich immer bedeutsamer geworden, da DL-Modelle heutzutage sehr komplex und Trainingsprozesse sehr zeit- und rechenintensiv sind. Unter TL versteht man das Wiederverwenden bereits vortrainierter neuronaler Netze für die Lösung neuartiger Probleme. Das initiale Training obliegt hierbei meist großen Unternehmen oder Institutionen. Dabei werden die erprobten Modelle sodann als Startpunkt genutzt und nur noch auf die neuen Probleme adaptiert, anstatt eigene Modelle von Grund auf neu zu trainieren. Anwender profitieren hier zeitlich, qualitativ und technisch. Zumeist sind architektonische Anpassungen in den hinteren Schichten der vortrainierten Modelle erforderlich, sodass sie sich für die Lösung der neuen Probleme eignen, wie Abbildung 2.14 veranschaulicht. Zudem ist ein gezieltes weitergehendes Training mit entsprechenden Daten notwendig. Inwieweit die neuen Daten auf die vortrainierten Modelle einwirken sollen, ist individuell zu erproben [Zhang et al., 2020, S. 554].

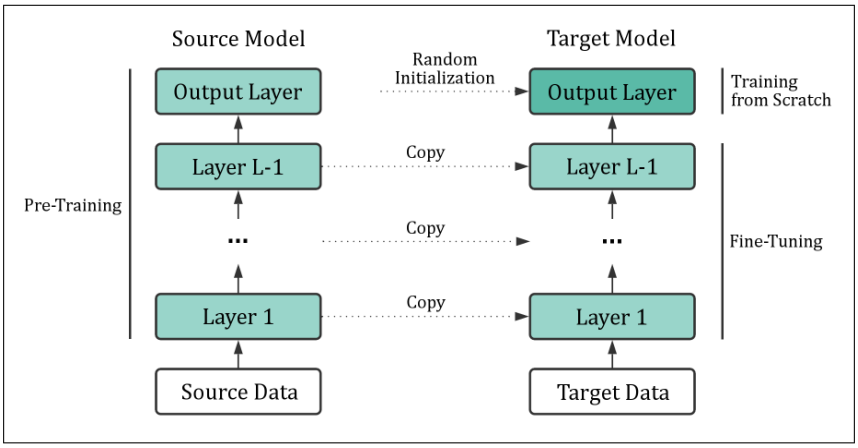

Abbildung 2.14: Fine-Tuning vortrainierter Modelle [Zhang et al., 2020, S. 555].

TL wird auch in dieser Arbeit genutzt. Einige Komponenten der bereits vorgestellten Architekturen, wie beispielsweise der Encoder oder auch der Decoder, können durch vortrainierte Modelle repräsentiert werden. Hier wird inhaltlich sowie kontextuell in den folgenden Kapiteln angeknüpft, da zunächst die Einführung weiterer NLP-Grundlagen erforderlich ist. Die angeführten Vorteile von TL können nichtsdestotrotz folgendermaßen zusammengefasst werden:

- Zeitersparnis durch Überspringen des initialen Trainings
- Qualitätsanstieg und Generalisierung durch Berücksichtigung massenhafter Daten
- Reduktion von hardwaretechnischen Anforderungen, Kosten und Stromverbrauch

3 Natural Language Processing

Natürliche Sprache wird auch als menschliche Sprache bezeichnet und ist historisch gewachsen. Sie verfolgt orthographische und grammatikalische Regeln auf Grundlage eines sprachabhängigen Wortschatzes. Die Sprachwissenschaft, auch Linguistik genannt, untersucht natürliche Sprache mithilfe verschiedener Methoden. NLP meint die maschinelle Verarbeitung natürlicher Sprache. Dabei werden Methoden der Linguistik unter anderem mit Methoden des Deep Learning verknüpft [Bird et al., 2009, S. 1]. Nicht selten ist eine Spracherkennung vorgeschaltet. NLP ist weiterhin in Natural Language Understanding (NLU) und Natural Language Generation (NLG) zu untergliedern [Bird et al., 2009, S. 27-28]. Diese Teilgebiete sind zugleich wesentliche Herausforderungen der ATS.

NLP-Aufgaben sind oftmals als Optimierungsprobleme zu verstehen. Lösungen sind demnach nicht eindeutig, also im mathematischen Sinne analytisch nicht lösbar. Dies wird in Hinblick auf die ATS deutlich, wenn man verschiedene Personen den gleichen Text zusammenfassen lässt. Zwar gleichen sich die als relevant identifizierten Informationen größtenteils, doch die Formulierungen sind mitunter sehr unterschiedlich. Folglich können auch mehrere Versionen korrekt sein.

Natürliche Sprache bedarf hinsichtlich maschineller Verarbeitung einer geeigneten mathematischen Form. Hierfür werden nachfolgend verschiedene Vorverarbeitungsschritte sowie Word Embeddings und Deep Language Representations vorgestellt. Der Anspruch auf Vollständigkeit entfällt aufgrund der Mächtigkeit des NLP, obgleich anknüpfende Inhalte bei Bedarf an den entsprechenden Stellen erläutert werden.

3.1 Vorverarbeitung

In nahezu allen Teilbereichen der Data Science stehen gewöhnlicherweise etliche Vorverarbeitungsschritte an, um die zu analysierenden Daten zu bereinigen, zu normalisieren und insgesamt in eine konsistente sowie geeignete Form zu bringen. Im NLP-Kontext sind indes komplexere Vorverarbeitungsschritte erforderlich, um die Daten für die eingeforderte mathematische Form zu präparieren [Bird et al., 2009, S. 86]. Eine Auswahl der in dieser Arbeit relevanten Schritte wird nachfolgend vorgestellt. In der Implementierung dieser chronologisch aufeinander folgenden Schritte spricht man auch von der NLP-Pipeline.

3.1.1 Textbereinigung

An erster Stelle der NLP-Pipeline steht die Textbereinigung, welche sich bezüglich eingehender Sequenzen insbesondere auf Sonderzeichen, Interpunktion sowie Klein- und Großschreibung konzentriert. Dabei ist es mitunter bereits herausfordernd, entsprechende Textstellen als solche zu identifizieren. Anschließend sind oftmals normalisierende Maßnahmen anzuwenden. Üblich ist beispielsweise das Entfernen von Sonderzeichen oder auch das Erzwingen von Kleinschreibung in allen eingehenden Texten [Bird et al., 2009, S. 107]. Weit verbreitet ist auch das Entfernen von Stoppwörtern. Dies sind Wörter, welche mutmaßlich der Allgemeinsprache zugehören, weshalb angenommen wird, dass sie keine entscheidende inhaltliche Bedeutung besitzen [Gambhir et al., 2016, S. 5]. Hier lässt sich jedoch keine allgemeingültige Aussage treffen, da die tatsächlich erforderlichen Maßnahmen sowohl von den Eigenschaften der Eingangsdaten als auch von den Besonderheiten der verwendeten Modelle und den verfolgten Zielen abhängen. Dabei ist die Datenexploration wiederkehrend und alternierend mit der Anpassung der Vorverarbeitung auszuführen. Der Anwender sollte hierbei ein Gefühl für die Daten und deren Besonderheiten entwickeln. Zudem bedarf es einem tiefgründigen Verständnis der geplanten Aufgaben, um beurteilen zu können, welche Vorverarbeitungsschritte tatsächlich relevant sind.

3.1.2 Textnormalisierung

In der weitergehenden Textnormalisierung wird sich vorrangig auf das Stemming und die Lemmatisierung konzentriert. Das Stemming führt eingehende Wörter auf ihre Grundformen zurück, indem bekannte Präfixe, Infixe und Suffixe eliminiert werden. Diese Grundformen sind nicht zwingend valide Wörter. Die Lemmatisierung hingegen berücksichtigt

die wortspezifischen Bedeutungen, um etwaige Flexionen in Deckung zu bringen und somit die linguistisch korrekten Grundformen zu bilden. Flexionen sind durch Konjugationen, Deklinationen oder auch Komparationen entstanden und natürlicher Bestandteil einer Sprache. Hierfür sind Wortbildungsregeln und ein Wortschatz erforderlich. Letzterer indiziert die eingehenden Wörter anhand ihrer Lemmata und ordnet sie entsprechend zu [Bird et al., 2009, S. 107-108]. Zwar ist die Lemmatisierung aufgrund des erforderlichen Kontextwissens durchaus komplexer als das Stemming, dafür sind ihre Ergebnisse erwartungsgemäß gehaltvoller. Ob und welche Methode zur Textnormalisierung herangezogen wird, hängt erneut von der anvisierten Aufgabe ab. Seien nun beispielhaft die Wörter {*spielen, spielst, spielte, gespielt*} gegeben, dann reduziert der Stemmer diese Wörter auf die Grundform *spiel*. Der Lemmatizer identifiziert hingegen die linguistisch korrekte Grundform *spielen*.

Natural Language Toolkit (NLTK) ist eine forschungsorientierte Python-Bibliothek, die etliche NLP-Module zur Verfügung stellt, darunter unter anderem verschiedenartige Stemmer [Bird et al., 2009, S. 13-14]. Stemmer, welche die englische Sprache unterstützen, scheinen bereits sehr ausgereift zu sein. Stemmer, welche die deutsche Sprache unterstützen, sind nicht nur knapp, sondern bedürfen zudem weitergehenden Testschritten, um deren tatsächliche Eignung zu prüfen. Die Stemmer nltk.stem.cistem und nltk.stem.snowball eignen sich potenziell für einen Einsatz mit deutscher Sprache [NLTK, 2020].

SpaCy ist eine eher praktisch orientierte Python-Bibliothek für verschiedenste NLP-Aufgaben. Hinsichtlich der deutschen Sprache eignen sich hier insbesondere die verfügbaren Lemmatizer. Dabei kann der Anwender zwischen verschiedenen vortrainierten Modellen wählen. Eigenschaften wie Sprache, Größe und zugrundeliegende Trainingsdaten sind transparent dokumentiert [SpaCy, 2021].

Die Wahl geeigneter Stemmer und Lemmatizer obliegt dennoch den subjektiven Präferenzen des jeweiligen Entwicklers. In jedem Fall sind hinreichende Tests durchzuführen, um die einzelnen Module zu erproben sowie individuelle Vor- und Nachteile zu identifizieren. Mit fortschreitender Entwicklung beweisen sich möglicherweise auch andere aufstrebende Bibliotheken [Bird et al., 2009, S. 108].

3.1.3 Tokenisierung

In der Tokenisierung werden Texte in logisch zusammengehörige Token zerlegt. Texte bestehen aus Sequenzen, welche wiederum aus Symbolen, also etwa Zeichen, Zeichenketten oder auch Ziffern bestehen. Token sind indes als Einheiten der Wort- oder Satzebene zu verstehen [Manning et al., 2008, S. 22-24].

Der einfachste Ansatz einer wortbasierten Tokenisierung besteht darin, den Text anhand von Leerzeichen und nicht-alphanumerischer Zeichen zu segmentieren. Dies ist jedoch nicht völlig obligatorisch und führt meist nicht zu einer verarbeitbaren Lösung, weshalb sprachabhängige Eigenarten berücksichtigt werden müssen. Typisch ist beispielsweise die Weiterbehandlung von vor- oder nachstehenden Klammern an den Token [Bird et al., 2009, S. 109-111].

Ob weiterhin auch Interpunktion berücksichtigt oder verworfen werden soll, ist hinsichtlich der anvisierten NLP-Aufgabe individuell zu entscheiden und zu erproben. Gleiches gilt für die Entscheidung, ob eingehende Texte roh oder vorverarbeitet hineingegeben werden.

Sei nun ein Satz gegeben, welcher keiner Textbereinigung und keiner Textnormalisierung unterzogen wird. Eine Tokenisierung, welche die Eigenschaften der deutschen Sprache sowie Interpunktion hinreichend berücksichtigt, würde die in Abbildung 3.1 visualisierte Menge von Token generieren.

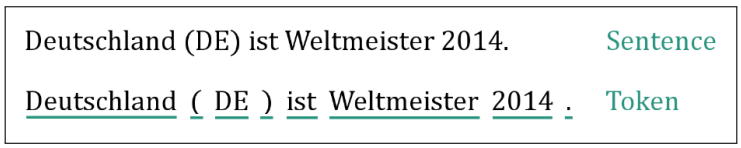

Abbildung 3.1: Tokenisierung eines beispielhaften Satzes.

Die Arbeit mit Texten erfordert bekanntermaßen eine geeignete mathematische Form. Die zeichenbasierten Token der Texte werden daher in einem Wortschatz (engl. Vocabulary) mithilfe numerischer Indizes kodiert. Hier ist es möglich, die Anzahl eindeutiger Token zu identifizieren oder gar seltene Token aus praktischen Gründen zu entfernen [Zhang et al., 2020, S. 311-312].

Die Python-Bibliotheken NLTK und SpaCy stellen entsprechende Tokenizer für eine möglichst schnelle Implementierung bereit. Beide sind überdies in einer für die deutsche Sprache ausgereiften Version verfügbar. Oftmals werden hierbei weitere Funktionalitäten mitgeliefert, darunter meist das Entfernen sprachbezogener Stoppwörter, die Lemmatisierung oder auch das Part-of-Speech-Tagging [Bird et al., 2009, S. 111].

3.2 Word Embeddings

Algorithmen können Texte bekanntermaßen nicht in ihrer Rohform verarbeiten. Texte bedürfen einer geeigneten mathematischen Form. Word Embeddings überführen Texte oder ganze Korpora hierfür in einen Vektorraum (engl. Vector Space), um Wörter syntaktisch, semantisch und insbesondere untereinander in Kontext zu bringen. Dabei wird ein Wortschatz benutzt, welcher die entsprechenden Vektoren, bestehend aus eindeutig kodierbaren ganzzahligen Werten, aufbaut [Karani, 2018]. Die Ableitung der Vektoren aus den Textdaten wird auch als Feature Extraction oder Feature Encoding bezeichnet. Insgesamt befindet man sich hier im Bereich des Language Modeling [Brownlee, 2019].

Word Embeddings werden üblicherweise noch vor der Entwicklung der ursprünglich anvisierten NLP-Aufgabe trainiert, weshalb ihnen ein unmittelbarer qualitativer Einfluss zugesprochen wird. Die entstehenden Modelle sind in der Folge schnell implementierbar. Weiterhin haben sie hiermit einen hohen skalierenden Effekt, da sie als Grundlage verschiedenster nachgelagerter NLP-Aufgaben eingesetzt werden können [Nitsche, 2019]. Word Embeddings können durch verschiedene mehr oder minder komplexe Ansätze realisiert werden. Diese werden nachfolgend vorgestellt, wobei stets verdeutlicht wird, wie der entsprechende Vektorraum aufgebaut und in Kontext gebracht wird. Dabei wird außerdem deutlich, dass sich die verschiedenen Ansätze nicht für jede NLP-Aufgabe eignen, sondern sie vielmehr einschränken. Obgleich nicht alle Ansätze für die ATS relevant sein werden, sind sie als Grundlage zu verstehen.

3.2.1 One-Hot-Encoding

One-Hot-Encoding (OHE) ist einer der einfachsten Ansätze, um Texte in einen Vektorraum einzubetten. Dabei werden allgemein kategorische Variablen in ein numerisches und somit mathematisch verarbeitbares Format gebracht [Karani, 2018]. Seien nun zwei gleichbedeutende Sätze gegeben. Eben jene Sätze sowie der entsprechende Wortschatz und die binär kodierten Vektoren sind in Abbildung 3.2 ersichtlich. Die Vektoren sind hierbei als Matrix zusammengefasst, wobei die Zeilen und Spalten anhand der Anfangsbuchstaben der Wörter kenntlich gemacht sind.

Versucht man nun, diese Vektoren in einem Vektorraum zu visualisieren, dann entspricht jeder Vektor einer eigenen Dimension. Dabei wird allerdings klar, dass keine dimensionsübergreifenden Projektionen existieren [Karani, 2018]. Dies bedeutet, dass die Wörter *gutes* und *schönes* genauso verschieden sind, wie die Wörter *heute* und *ist*. Dies ist offensichtlich falsch. OHE ist dennoch als Grundlage zu verstehen, wobei etwaige Probleme innerhalb der nachfolgenden Ansätze weiterbehandelt werden.

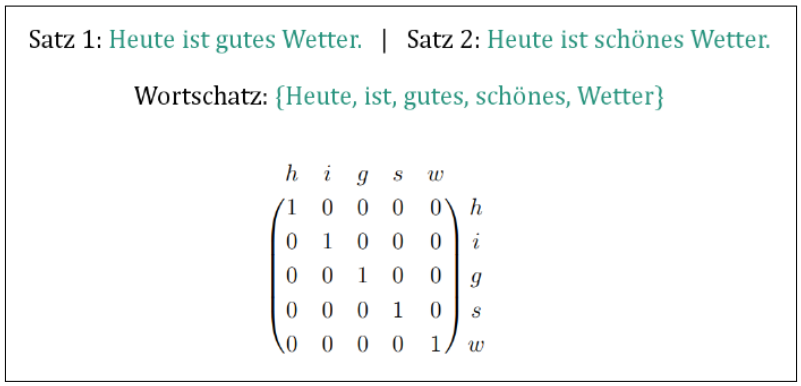

Abbildung 3.2: One-Hot-Encoding mit zwei beispielhaften Sätzen.

3.2.2 Bag-of-Words

Bag-of-Words (BOW) realisieren die Feature Extraction, indem entsprechende Modelle das Aufkommen von in einem Wortschatz definierter Wörter über eine Vielzahl von Texten zählen. Dabei ist allerdings ein gewisser Informationsverlust zu erwarten, da beispielsweise die Reihenfolge der Wörter nicht berücksichtigt wird [Raschka et al., 2019, S. 262]. Tabelle 3.1 zeigt beispielhaft eine Matrix eines solchen BOW-Modells.

In den Zeilen sind die betrachteten Texte indiziert, in den Spalten hingegen die Wörter des frei erfundenen Wortschatzes. Die Matrix hat demnach eine daran orientierte fixe Größe. In den Zellen ergeben sich somit die Häufigkeiten der Wörter, bezogen auf ihr Aufkommen in den einzelnen Texten [Brownlee, 2019].

ID	**Heute**	**ist**	**schönes**	**Wetter**
Text 1	2	8	1	3
Text 2	1	3	0	2
Text 3	0	4	1	4

Tabelle 3.1: Bag-of-Words mit einem beispielhaften Wortschatz [Huilgol, 2020].

Nachteilig ist dieser Ansatz hinsichtlich der ATS insbesondere dadurch, dass die Reihenfolge der Wörter nicht berücksichtigt wird. Wie in Tabelle 3.1 ersichtlich wurde, lassen sich aus der Matrix keine Informationen über die Semantik oder Grammatik rekonstruieren. Aus technischer Sicht würde die Matrix bei steigender Wortschatzgröße nicht nur mitwachsen, sondern zudem viele Null-Einträge enthalten [Huilgol, 2020]. Vorteilig ist dieser Ansatz hingegen für eher schlichtere NLP-Aufgaben. Hier sei die Klassifikation von Dokumenten als mögliches Einsatzgebiet genannt.

3.2.3 Skip-Gram-Model

Ein weiterer frequenzbasierter Ansatz besteht in sogenannten Skip-Gram-Modellen. Diese unterliegen der Annahme, dass sich der Kontext eines gegebenen Wortes in Form von einer Textsequenz generieren lässt. Sei hierfür nun beispielhaft und gleichermaßen abstrakt die Sequenz $\{a,\ b,\ c,\ d,\ e\}$ gegeben. Zudem sei c das Zielwort und die lokale Fenstergröße zwei. Ein Skip-Gram-Modell modelliert die bedingten Wahrscheinlichkeiten für die vor- und nachstehenden Kontextwörter [Zhang et al., 2020, S. 640]. Hierfür gilt die folgende Formel:

$$P(a, b, d, e \mid c)$$

Gemäß der weitergehenden Annahme, die Kontextwörter ließen sich auf Grundlage eines gegebenen Zielwortes unabhängig voneinander generieren, kann die Formel wie folgt umgeschrieben werden:

$$P(a \mid c) \cdot P(b \mid c) \cdot P(d \mid c) \cdot P(e \mid c)$$

Sei darüber hinaus abstrakter Wortschatz gegeben. Dabei erfordert jedes darin enthaltene Wort zwei mehrdimensionale Vektoren. Einen, um das Wort als Zielwort zu evaluieren, und einen, um das Wort in den unterschiedlichen Kontexten einzuordnen. Mithilfe dieser Vektoren können die bedingten Wahrscheinlichkeiten des entsprechenden Modells trainiert werden. Autark ist jedoch auch dieser Ansatz ungeeignet für die ATS [Zhang et al., 2020, S. 641].

3.2.4 Word2Vec

Der Ansatz des Word2Vec (W2V) kombiniert BOW-Modelle mit Skip-Gram-Modellen, um die Nachteile des OHE weitergehend aufzuarbeiten. Dabei werden BOW-Modelle jedoch in kontinuierlicher Form genutzt. Diese funktionieren in umgekehrter Weise zu den Skip-Gram-Modellen. Damit ist folglich eine beidseitige Herangehensweise möglich. Der Kontext kann also aus einem gegebenen Zielwort und das Zielwort wiederum aus einem gegebenen Kontext ermittelt werden [Zhang et al., 2020, S. 644].

Die entsprechenden bedingten Wahrscheinlichkeiten können gemäß der nachstehenden Formeln modelliert werden. Dabei werden Skip-Gram-Modelle (erstgenannt) den BOW-Modellen (zweitgenannt) anhand des oben genannten Beispiels gegenübergestellt.

$$P(a, b, d, e \mid c) \qquad P(c \mid a, b, d, e)$$

W2V-Modelle können mithilfe neuronaler Netze trainiert werden. Dies befähigt sie, Zusammenhänge zwischen Wörtern zu erlernen. Hierbei werden Distanzen minimiert, die aus den zuvor beschriebenen Vektoren hervorgehen. Dieser Ansatz eignet sich in der Folge etwa dazu, Synonyme für gegebene Wörter zu bestimmen. ATS-Modelle, welche den W2V-Ansatz verfolgen, konnten sich in der Vergangenheit bereits in akzeptablem Maße beweisen [Karani, 2018].

Abbildung 3.3 zeigt die beispielhaften Projektionen von etwa 10.000 Wörtern, welche mit dem Embedding Projektor von TensorFlow und einer entsprechenden Hauptkomponentenanalyse generiert wurden. Dabei ist das Ergebnis der Suche nach dem Wort *play* zu sehen. Es ist erkennbar, dass bedeutungsähnliche Wörter kürzere Distanzen aufweisen.

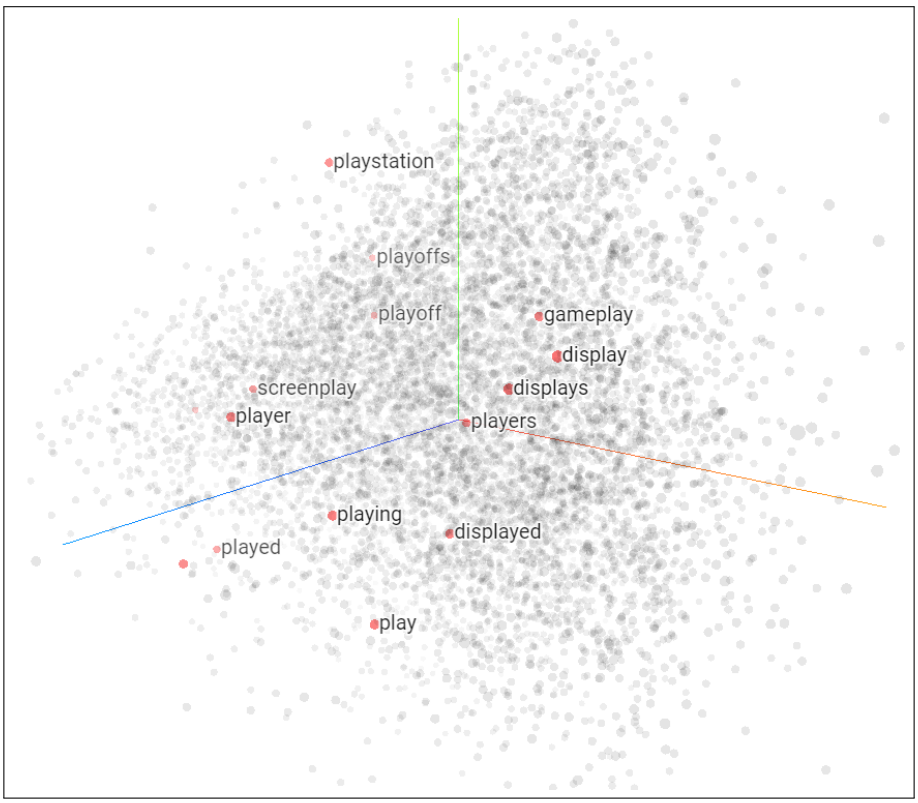

Abbildung 3.3: Word2Vec mit dem Embedding Projector von TensorFlow.

3.2.5 Byte-Pair-Encoding

Byte-Pair-Encoding (BPE) ist ein Algorithmus zur Datenkompression. Hierfür werden zusammenhängende Bytes mit einem Byte ersetzt, welches nicht in den sonstigen Daten auftritt [Nitsche, 2019, S. 24]. Sei folgendes Beispiel gegeben:

$$aaabcaacab = \boldsymbol{ZY}c\boldsymbol{Z}c\boldsymbol{Y}$$

$$\text{mit } \boldsymbol{Z} = aa \text{ und } \boldsymbol{Y} = ab$$

BPE eignet sich insbesondere dafür, seltene Wörter zu berücksichtigten. In diesen verbirgt sich mitunter eine nicht zu vernachlässigende Bedeutung. Aufgrund der kodierenden Funktion wird BPE auch als Subword Embedding bezeichnet. Aufgrund der komprimierenden Funktion ist weiterhin von kleineren Modellgrößen auszugehen [Nitsche, 2019, S. 24]. ATS-Modelle, welche den W2V-Ansatz verfolgen, konnten sich in der Vergangenheit ebenfalls in akzeptablem Maße beweisen.

3.2.6 GloVe

Global Vectors for Word Representation (GloVe) ist ein weiterer Algorithmus zur Einbettung von Texten in einen Vektorraum, welcher syntaktische und semantische Bedeutungen repräsentiert [Penninglon et al., 2014, S. 1].

Hierfür wird sich einer Matrix bedient, welche das gemeinsame Aufkommen der im Korpus enthaltenen Wörter paarweise gegenüberstellt. Sie wird daher auch als Word-Word-Co-Occurrence-Matrix bezeichnet und besitzt eine maximale Komplexität von $O(|V|^2)$, wenn V die Wortschatzgröße ist. GloVe sieht dabei vor, die Matrix unter Nutzung lokaler Kontextfenster global zu faktorisieren [Penninglon et al., 2014, S. 2].

Hierbei werden sehr große Matrizen durch mehrere niederrangige Matrizen approximiert. Dies ermöglicht es, die verborgenen statistischen Informationen des zugrundeliegenden Korpus anhand linearer Substrukturen zu explorieren und letztlich zu extrahieren. Dabei werden lokale Kontextfenster verwendet, wie sie bei verschiedenen oben genannten Ansätzen bereits beschrieben wurden, beispielsweise bei W2V [Nitsche, 2019, S. 24].

GloVe ist praktisch als vortrainiertes Modell verfügbar. Dies basiert auf vier verschiedenen Korpora mit insgesamt etwa 55 Milliarden Token und einer durchschnittlichen Wortschatzgröße von etwa 1,5 Millionen Token. Im Training werden hierbei in der Matrix nur von Null verschiedene Werte berücksichtigt. Die mittlere quadratische Abweichung wird zudem als Fehlerfunktion ausgewählt. Das initiale Training ist zwar sehr rechenintensiv, dafür allerdings im Sinne des TL a priori nur einmalig auszuführen. NLP-Aufgaben, darunter auch die ATS, profitierten stark vom Erfolg des GloVe-Ansatzes. Trotzdem ist der Einsatz stets sorgfältig zu evaluieren, da innerhalb der modellinternen Vorverarbeitung die zugrundeliegenden Korpora beispielsweise auf Kleinschreibung forciert wurden. Dies müsste bei nachstehenden NLP-Aufgaben ebenfalls berücksichtigt werden [Penninglon et al., 2014, S. 6-9].

3.3 Deep Language Representations

Die beschriebenen Word Embeddings verfolgen teils sehr abstrakte Ansätze, welche sich mitunter nur auf sehr wenige Teilbereiche des NLP adaptieren lassen oder entscheidende Nachteile aufweisen. Zudem unterliegen sie meist statistischen und somit rechentechnischen Limitationen.

ATS-Modelle erfordern daher weitergehende Deep Language Representations, um den Anforderungen an das NLU und die NLG gerecht zu werden. Dabei werden die ursprünglichen Ansätze durch umfangreichere und komplexere neuronale Netze ersetzt. Diese werden fortan als Language Models bezeichnet. Nur wenige marktpräsente Akteure sind hierbei dazu fähig, geeignete Architekturen zu konzipieren und entsprechende Modelle von Grund auf neu zu trainieren. Dies wurde in der Vergangenheit bereits getan und veröffentlicht, sodass gemäß TL nur noch ein weitergehendes Fine-Tuning nötig ist. Dabei profitierten nahezu alle nachstehenden NLP-Aufgaben qualitativ von diesen Fortschritten. Architekturen, welche als Deep Language Representations zu verstehen sind und eine ATS begünstigen, werden nachfolgend offengelegt [Nitsche, 2019, S. 25].

3.3.1 ELMo

Zuerst sei Embeddings from Language Models (ELMo) als Deep Contextualized Word Representation genannt. Es wurde vom Allen Institute for Artificial Intelligence entwickelt und 2018 veröffentlicht. ELMo modelliert dabei einerseits komplexe Wortcharakteristiken wie Syntax und Semantik, andererseits aber auch den linguistischen Kontext eben dieser Wörter. Es ist als vortrainiertes Modell verfügbar und konnte nach der Veröffentlichung insbesondere NLU-Aufgaben begünstigen [Peters et al., 2018, S. 1].

ELMo arbeitet mit wortbezogenen kontextbasierten Vektoren, welche aus zwei bidirektionalen Language Models abgeleitet und trainiert werden. Diese bestehen wiederum aus zwei gekoppelten LSTM-Schichten. Der umfangreiche Korpus an Textdaten wird dabei schichtweise vorwärts und rückwärts durch die Architektur propagiert. Somit sind kontextabhängige Informationen verfolgbar und die Bedeutungen der Wörter erlernbar. Aufgrund der zeichenbasierten Funktionsweise kann ELMo sogar Wissen über Wortformen erlernen. Im Vergleich zu W2V oder etwa GloVe werden die Vektoren nicht tokenbasiert berechnet, sondern mithilfe einer Funktion, welche stets eine Sequenz berücksichtigt. Wörter können daher in verschiedenen Kontexten verschiedene Vektoren besitzen. Abbildung 3.4 visualisiert dies [Peters et al., 2018, S. 2-3].

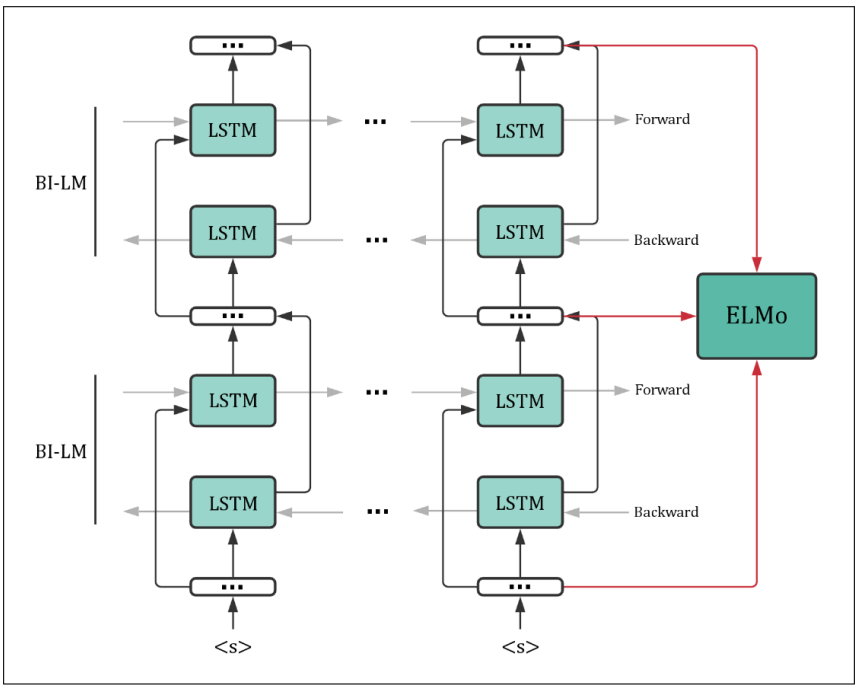

Abbildung 3.4: Architektur und Funktionsweise von ELMo [Irene, 2018].

3.3.2 GPT

OpenAI arbeitet mit dem Generative Pre-Trained Transformer (GPT) als Language Model in der nunmehr dritten Generation, veröffentlicht es allerdings nicht. Zu Forschungszwecken stellen sie dennoch eine schmalere Version des Modells bereit. OpenAI zeigt insbesondere mit dem GPT, dass Language Models mithilfe der sogenannten Next Word Prediction besonders gut im Sinne des Unsupervised Learning trainiert werden können. Es basiert zudem auf der bereits bekannten Transformer-Architektur und begünstigt somit verschiedene nachgelagerte NLP-Aufgaben, ohne entsprechende gelabelte Daten jemals gesehen zu haben. Dabei ist das GPT meist besser als konkurrierende Modelle, welche auf den aufwendig zusammenzustellenden gelabelten Daten basieren, auch im sogenannten Zero-Shot-Transfer. Einer der größten Erfolge ist hierbei, dass das GPT gemäß NLG selbst einen Nachrichtenartikel verfassen konnte [Radford et al., 2019, S. 1].

Die zugrundeliegenden Daten des GPT-Modells bestehen aus 40GB an Texten, welche aus acht Millionen Internetseiten kollektiert wurden und zugleich bestimmten Qualitätsanforderungen gerecht werden. Hier nimmt OpenAI unter anderem an, dass Reddit-Beiträge mit einem ausreichend hohen positiven Karmastand eine erhöhte Qualität versprechen [Radford et al., 2019, S. 3].

Im mathematischen Sinne wurde im Kontext der ATS bislang meist die Wahrscheinlichkeit der Ausgabesequenz unter einer gegebenen Eingabesequenz modelliert. Language Models wie das GPT streben hingegen ein allgemeineres Sprachverständnis an, um darüber hinaus eine Vielzahl an NLP-Aufgaben unterstützen zu können. Daher modelliert das GPT gemeinhin die Wahrscheinlichkeit $P(output \mid input, task)$ [Radford et al., 2019, S. 2]. Aufgrund der eingeschränkten Verfügbarkeit seitens OpenAI entfallen an dieser Stelle weitere Ausführungen.

3.3.3 BERT

Zudem sei BERT als mögliches Modell zur Deep Language Representation genannt. Es wurde von Forschern der Google AI Language entwickelt und 2019 veröffentlicht. BERT basiert auf ebenfalls der bereits bekannten Transformer-Architektur, wobei unstrukturierte Textdaten mithilfe links- und rechtsseitiger Kontextfenster bidirektional angelernt werden. Dabei entstammen die Daten verschiedenartigen NLP-Aufgaben, sodass BERT größtenteils aufgabenunabhängig einsetzbar ist. Im anschließenden Fine-Tuning werden die vortrainierten modellinternen Parameter zunächst initialisiert und wiederum mithilfe von gelabelten Daten angepasst. BERT zeichnet sich weiterhin dadurch aus, dass er höchst anpassungsfähig ist und sich die vortrainierte Architektur nur minimal von der weitertrainierten Architektur unterscheidet [Devlin et al., 2019, S. 1-3].

Nun wird die Architektur von BERT umfassender untersucht, indem die einleitenden Worte aufgegriffen und theoretisch ergründet werden. BERT ist prinzipiell als Encoder zu verstehen, welcher durch einen mehrschichtigen bidirektionalen Transformer repräsentiert wird. Dabei werden die Schichten durch Transformer-Module repräsentiert. Diese verfügen über entsprechende Attention-Mechanismen, um kontextuelle Informationen zu erkennen. Weiterhin werden Sequenzen nicht nur einseitig betrachtet, sondern beidseitig, also bidirektional. Somit werden wortbezogene Kontexte gelernt, was insbesondere zu einem tieferen Verständnis der Sprache seitens BERT führt. Dies ist ein großer Vorteil gegenüber bisheriger Architekturen, inklusive ELMo und GPT, welche Sequenzen zumeist nur einseitig betrachten konnten, also von links nach rechts oder andersherum. Aufgrund der qualitativen Vorteile von BERT wird es umfänglicher als die ausgewählten Vorgänger und Konkurrenten analysiert [Devlin et al., 2019, S. 3].

Um BERT entsprechend vortrainieren zu können, werden Masked Language Models (MLM) genutzt. Bevor die Textdaten in die Architektur geführt werden, werden etwa 15% der Wörter durch ein [MASK]-Token ersetzt. Daraufhin versucht das Modell, die maskierten Token auf Grundlage der umliegenden nicht-maskierten Token vorherzusagen (engl. Next Word Prediction). Da die zu erwartenden Wörter bekannt sind, kann über eine Klassifikationsschicht ein Lerneffekt erzielt werden, indem entsprechende Matrizen dimensional transformiert und wertmäßig aktualisiert werden. Abbildung 3.5 zeigt die beschriebene Architektur [Devlin et al., 2019, S. 4-5].

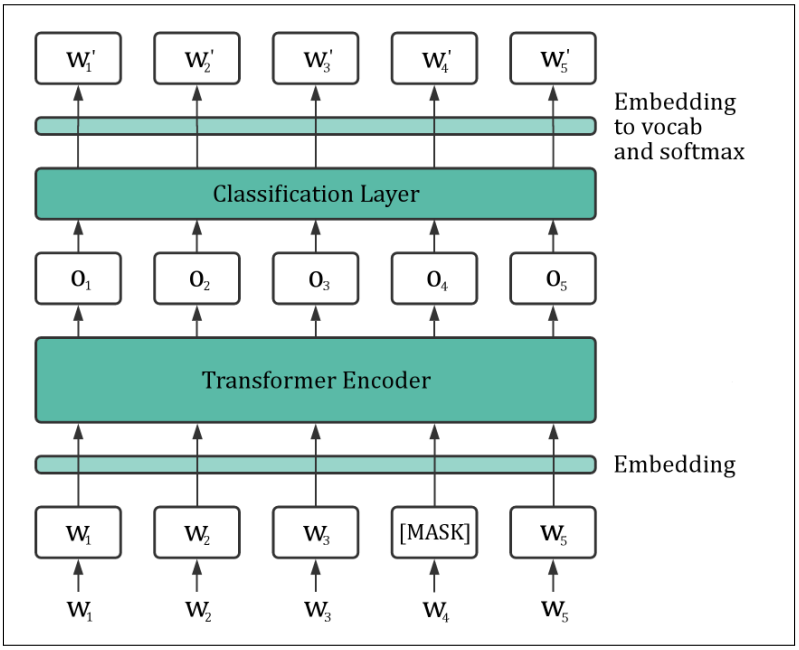

Abbildung 3.5: Architektur von BERT mit MLM [Devlin et al., 2019, S. 3].

Um zudem weitergehende Informationen auf Satzebene zu erhalten, werden dem Training paarweise Sätze zugeführt. Seien A und B zwei solcher Sätze, dann ist B zu 50% ein auf A folgender Satz und zu 50% ein zufälliger Satz des Korpus. Dies wird erneut im Sinne einer Klassifikation gelernt (engl. Next Sentence Prediction). Wie BERT die eingehenden Sequenzen einbettet und repräsentiert, wird in Abbildung 3.6 deutlich. Dabei handelt es sich um die Summe der Token Embeddings, Segment Embeddings und Position Embeddings. Hierbei kann entschieden werden, ob Klein- und Großschreibung berücksichtigt werden soll [Devlin et al., 2019, S. 3-5].

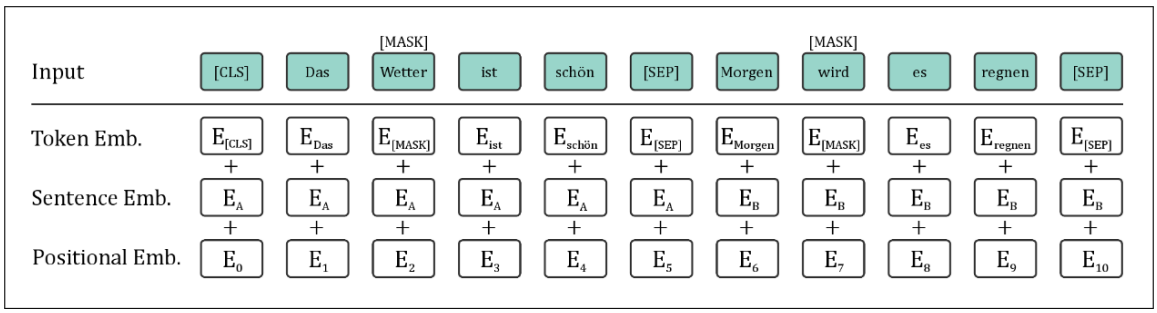

Abbildung 3.6: Repräsentation von Textdaten mithilfe von BERT [Devlin et al., 2019, S. 3].

BERT wurde initial in zwei verschiedenen Versionen veröffentlicht: BASE und LARGE. Die veröffentlichten Modelle wurden von den Forschern der Google AI Language auf zwei verschiedenen Korpora mit insgesamt über drei Millionen Wörtern trainiert. Sei L die Anzahl der Schichten (oder: Transformer-Module), H die Größe der verborgenen Schichten und A die Anzahl der Self-Attention-Heads, dann sind die Versionen wie folgt definiert [Devlin et al., 2019, S. 3-5].

$$\text{BERT}_{\text{BASE}} : (L = 12, H = 768, A = 12) \text{ mit } 110\text{M Parametern}$$

$$\text{BERT}_{\text{LARGE}} : (L = 24, H = 1024, A = 16) \text{ mit } 340\text{M Parametern}$$

3.3.4 XLM

Cross-Lingual Language Model RoBERTa (XLM-R) ist ein weiteres Language Model, welches 2019 von Facebook AI veröffentlicht wurde und der Architektur von BERT nahekommt. Es basiert demnach ebenfalls auf der bereits bekannten Transformer-Architektur. Es wird weiterhin über die Vorhersage absichtlich versteckter Textabschnitte unüberwacht trainiert. XLM-R zeichnet sich insbesondere durch die multilinguale Funktionsweise aus. Hierfür wurde es auf mehr als zwei Terabyte an Textdaten vortrainiert, darunter Texte aus 100 verschiedenen Sprachen. Die Next Sentence Prediciton entfällt hingegen im Vergleich zu BERT, während die LR deutlich erhöht wird. Letztlich ist von XLM-R durch die entsprechenden Anpassungen eine qualitative Verbesserung in verschiedenen NLP-Aufgaben gegenüber der multilingualen Version von BERT zu erwarten. Offensichtlich werden sprachübergreifende verborgene Strukturen erlernt, welche sich gegenseitig begünstigen. Darüber hinaus verwendet XLM-R einen sprachübergreifenden Tokenizer. XLM-R fordert Rechnern allerdings auch eine höhere Leistung ab [Conneau et al., 2020].

3.3.5 BART

Denoising Sequence-to-Sequence Pre-training for NLG (BART) ist eine weitere Architektur der Facebook AI ais dem Jahre 2019, welche sich insbesondere für das Vortraining von Sequence-to-Sequence-Modellen eignet. Hierfür wird eingehender Text mithilfe einer Rauschfunktion verändert. Das Modell versucht daraufhin, den ursprünglichen Text zu rekonstruieren, um unüberwacht einen Lernfortschritt zu erzielen. BART generalisiert bisherige Architekturen wie BERT oder auch GPT, weshalb es sich weiterhin um eine Transformer-Architektur handelt. Die besten Ergebnisse erzielte BART in der jüngsten Vergangenheit bei NLG-Aufgaben wie der ATS, sowohl mono- als auch multilingual. Dabei schließt BART qualitativ nicht nur zu XLM-R auf, sondern übersteigt sie zu Teilen sogar [Lewis et al., 2019].

3.4 Metriken

Weiterhin sind ausgewählte Metriken offenzulegen, mit denen die Qualität der ATS gemessen werden kann: Recall-Oriented Understudy for Gisting Evaluation (ROUGE) und Bilingual Evaluation Understudy (BLEU). In der Wissenschaft werden ATS-Modelle meist mithilfe des ROUGE-Scores evaluiert und verglichen. Dabei erfordern diese Metriken eine Referenzzusammenfassung (REFZ) zu jeder Systemzusammenfassung (SYSZ), welche maschinell generiert wurde. Dies ist unabhängig davon, ob das Training überwacht oder unüberwacht durchgeführt wird. Allgemein unterliegen diese Metriken der Herausforderung, dass es für einen gegebenen Text keine objektiv beste Zusammenfassung gibt. Folglich können verschiedene SYSZ oder sogar REFZ gleich gut sein. Dies ist statistisch sehr schwer zu bewerten, zumal selbst Menschen aufgrund ihrer subjektiven Bewertungsweise nicht einheitlich definieren können, welche Faktoren für eine gute Zusammenfassung stehen. Weiterhin werden in den Metriken menschliche Bewertungsfaktoren wie beispielsweise Lesbarkeit nicht berücksichtigt [Lemberger, 2020].

3.4.1 ROUGE

ROUGE kann zunächst mithilfe folgender Kennzahlen weitergehend differenziert werden: Recall, Precision und Measure. Dabei quantifiziert der Recall-Score den Anteil der sowohl in REFZ als auch in SYSZ vorkommender Wörter gemäß folgender Formel:

$$\frac{\text{Anzahl übereinstimmender Wörter}}{\text{Anzahl der Wörter in der REFZ}}$$

Sei hierfür verkürzt „Der Sommer war sehr warm“ als REFZ und „Der Sommer war wieder sehr warm“ als SYSZ gegeben. Dann gilt: Recall $= \frac{5}{5} = 1.0$. Trotzdem sollen SYSZ nicht unendlich lang werden, um alle Wörter der REFZ abzudecken, sondern weiterhin den eigentlichen Sinn einer Zusammenfassung erfüllen. Hier quantifiziert der Precision-Score den Anteil der tatsächlich relevanten Wörter gemäß folgender Formel:

$$\frac{\text{Anzahl übereinstimmender Wörter}}{\text{Anzahl der Wörter in der SYSZ}}$$

Wie man sieht, ändert sich nur der Nenner. Im oben genannten Beispiel gilt somit: Precision $= \frac{5}{6} = 0.8\overline{3}$. Nimmt man ferner „Der letzte Sommer war wieder sehr warm und trocken“ als neue SYSZ an, dann reduziert sich der Precision-Score aufgrund der erhöhten Anzahl unrelevanter Wörter wie folgt: Precision $= \frac{5}{9} = 0.5\overline{5}$. Weiterhin kann der Measure-Score als gewöhnlicher F-Score verstanden und interpretiert werden. Er ergibt sich als harmonisches Mittel zwischen dem Recall und der Precision, womit er beide Scores berücksichtigt [Lin, 2004, S. 1-3].

ROUGE wird allgemein auch als ROUGE-N geschrieben, wobei das N bestimmt, ob obige Kennzahlen auf Grundlagen von Uni-, Bi- oder Trigrammen berechnet werden sollen. Im genannten Beispiel wurden also die ROUGE-1 Recall-, Precision- und Measure-Scores berechnet. Zudem existieren Ansätze, welche die Longest Common Subsequence (LCS) verfolgen. Diese werden hier jedoch vernachlässigt.

Trotz oder gerade wegen der wissenschaftlichen Verbreitung des ROUGE-Scores kommt immer mehr Kritik auf. Demnach kann ROUGE beispielsweise nicht zwischen verschiedenen aber bedeutungsähnlichen Wörtern unterscheiden. Dies führt tendenziell zu einer schlechteren Bewertung, obgleich ein gegebener Text etwa in einer entsprechenden SYSZ präzise zusammengefasst wurde. Außerdem wird den Texten zur Berechnung des ROUGE-Scores Kleinschreibung abverlangt, unabhängig von den vorgeschalteten Modellen. Die Bewertung geschieht also eher auf syntaktischer als auf semantischer Basis. Aufgrund der bereits genannten weitreichenden Nutzung des ROUGE-Scores und der damit gegebenen Vergleichbarkeit kommt er trotzdem in dieser Arbeit zum Einsatz [Lin, 2004, S. 5].

3.4.2 BLEU

BLEU kommt der Funktionsweise von ROUGE weitestgehend gleich. Demnach repräsentiert der Score ebenfalls die Ähnlichkeit längendefinierter N-Gramme. Dabei wird weiterhin für jede SYSZ eine entsprechende REFZ gefordert. Beide Metriken funktionieren indes sprachunabhängig. Im Unterschied zu ROUGE führt BLEU einen multiplikativen Bestrafungsterm ein, um zu kurze SYSZ zu entwerten. Dies ist nicht notwendig, wenn die SYSZ länger ist als die REFZ. Der Precision-Score berücksichtigt dies bereits [Papineni et al., 2002, S. 5].

Obgleich der BLEU-Score primär die Bewertung von Übersetzungen unterstützt, eignet er sich gewissermaßen auch für ATS-Aufgaben. Zuletzt konnte wissenschaftlich bewiesen werden, dass der BLEU-Score recht gut mit menschlichen Bewertungen korreliert. Aufgrund ähnelnder Nachteile zu denen des ROUGE-Score und fehlender wissenschaftlicher Vergleichbarkeit wird der BLEU-Score in dieser Arbeit nicht verwendet [Papineni et al., 2002, S. 6-7].

4 Automatic Text Summarization

Unter Kenntnis der theoretischen Grundlagen werden die Ausführungen zur ATS in diesem Kapitel ausgeweitet, basierend auf den einordnenden Worten aus der Einleitung. Dabei werden die grundlegenden Informationen vollständigerweise zusammengefasst und mithilfe des neu gewonnenen Wissens nochmalig eingeordnet. Darüber hinaus wird insbesondere der Forschungsstand vertieft, vergleichbare Arbeiten analysiert und daraufhin selbst eine Architektur konzipiert.

4.1 Typisierung von Systemen

Es ist bereits bekannt, dass die ATS verschiedenartig typisiert werden kann. Dies wird in der nachstehenden Liste ersichtlich. Dabei wird die Typisierung der in dieser Arbeit behandelten ATS entsprechend der Zielsetzung markiert. Faktoren, welche die entsprechenden Entscheidungen herbeiführen, werden anschließend hinreichend erläutert [Gambhir et al., 2016, S. 5].

- Extractive vs. **Abstractive**
- **Generic** vs. Query-Focused
- **Single-Document** vs. Multi-Document
- **Supervised** vs. Unsupervised
- Mono-/ **Multi-**/ Cross-**Lingual**

Zunächst muss zwischen einer extraktiven und einer abstraktiven Methodik differenziert werden. Dabei ist zugleich zu definieren, ob Zusammenfassungen eher allgemein oder abfragebasiert generiert werden sollen. Dies wird von den individuellen Anforderungen an die einzusetzende ATS bedingt und bestimmt. Von weiteren Erklärungen zu den genannten Methodiken wird an dieser Stelle abgesehen, weil die entsprechenden Grundlagen entweder bereits erklärt wurden oder im weiteren Verlauf der Arbeit nicht mehr relevant sein werden.

Bevor ein ATS-System tatsächlich entwickelt werden kann, sind weitere Eigenarten zu identifizieren. Dies betrifft fortan insbesondere die Beschaffenheit der verfügbaren Daten, welche als Entscheidungsfaktor in die Typisierung der ATS eingehen. Daher ist zu ergründen, ob Zusammenfassungen auf nur einem oder mehreren Dokumenten basieren. Zudem ist anhand der verfügbaren Daten abzulesen, ob das Training überwacht oder unüberwacht stattfinden kann. Zuletzt ist ein ATS-System durch die Sprache(n) der zugeführten Daten limitiert [Gambhir et al., 2016, S. 5].

Technisch sei zusammengefasst, dass bei der ATS eine Eingabesequenz $x = [x_1, ..., x_n]$ in eine Ausgabesequenz $y = [y_1, ..., y_m]$ überführt wird, wobei n die Eingabelänge und m die Ausgabelänge ist. Dabei gilt stets $m < n$ mit dem Ziel $P(y \mid x)$. Die wesentlichen Herausforderungen bestehen dabei aus dem NLU und der NLG [Nitsche, 2019, S. 32-33].

4.2 Vertiefung des Forschungsstands

Nun wird der einleitend bereits umrissene Forschungsstand für den soeben eingegrenzten ATS-Typ vertieft. Neben vielversprechenden Ansätzen der letzten Jahre, welche mitunter auf RNN, LSTM und RL basierten, etablierten sich im SOTA seit 2018 bekanntermaßen Transformer. Nachfolgend wird daher beschrieben, wie Transformer bislang im Kontext der ATS eingesetzt wurden und wie Verbesserungen erzielt werden können. Hierfür werden die eingangs aufgelisteten vergleichbaren Arbeiten nun mit dem neu gewonnenen Wissen ausführlicher thematisiert und eingeordnet.

Yang et al. übertrafen 2019 den SOTA im Bereich der abstraktiven ATS mithilfe einer Encoder-Decoder-Architektur, wobei der Encoder durch einen vortrainierten BERT und der Decoder durch einen zufällig initialisierten Transformer repräsentiert wurde. Eine wesentliche Erkenntnis hierbei war die Multifunktionsfähigkeit von BERT zur NLU und zur NLG [Yang et al., 2019].

Nitsche befasste sich 2019 ebenfalls mit abstraktiver ATS. Dabei verwendete er in seiner Architektur ebenfalls BERT, jedoch in einer multilingual vortrainierten Version. Damit gelang ihm der Versuch, das Modell trotz mangelnder Ressourcen im deutschsprachigen NLP-Bereich auf die deutsche Sprache zu adaptieren. Aufgrund nicht verfügbarer Daten lässt sich keine weitergehende Aussage über die Qualität seiner Ergebnisse treffen [Nitsche, 2019].

Rothe et al. verfolgten 2020 eine Transformer-to-Transformer-Architektur zur ATS. Dabei werden die entsprechenden Encoder und Decoder in jedem Experiment durch vortrainierte Transformer repräsentiert, darunter beispielsweise BERT-to-BERT, XLM-R-to-XLM-R oder sogar BERT-to-XLM-R. Hierbei kamen Rothe et al. zu der Erkenntnis, dass monolinguales Fine-Tuning zu besseren Resultaten führt. Zugleich wird deutlich, dass multilinguales Vortraining profitabel sein kann. Die folgenden ROUGE-Scores können als SOTA registriert werden: R-Recall: 15.96, R-Precision: 10.34, R-Measure: 12.22 [Rothe et al., 2020].

In den soeben vertieften Publikationen wird ersichtlich, dass gemäß TL vortrainierte Transformer sowohl als Encoder als auch als Decoder genutzt werden können, stets unter gegebener Austauschbarkeit. Die sprachtechnische Adaption hingegen wurde bislang nur unzulänglich oder nicht-öffentlich erprobt. Daher wird in dieser Arbeit eine Architektur zur ATS eingeführt, welche die Fortschritte aufgreift sowie vorhandene Unzulänglichkeiten verbessert. Hierfür werden Transformer-Kombinationen mit besonderem Fokus auf die deutschsprachige Adaption erprobt. Gemäß dem SOTA werden BERT, XLM-R und BART ausgewählt, nicht hingegen ELMo und GPT.

4.3 Konzeption einer Architektur

Mit dem bisherigen Wissen wird nun eine Architektur konzipiert. Diese ist als Sequence-to-Sequence-Transformer-Modell zu verstehen. Dabei wird sowohl der Encoder als auch der Decoder durch ein eigenständiges gemäß TL vortrainiertes Modell repräsentiert. Aufgrund der bereits beschriebenen Grundlagen wird nun die weitergehende Konfiguration der Architektur definiert. Hierbei gibt es unter anderem die folgenden beiden Möglichkeiten, um den Encoder zum NLU und den Decoder zur NLG zu initialisieren [Rothe et al., 2020, S. 2].

Einerseits ist es möglich, den Encoder und den Decoder jeweils mit einem autarken Modell zu initialisieren, beispielsweise den Encoder mit BERT und den Decoder mit GPT. Andererseits ist es aber auch möglich, sowohl den Encoder als auch den Decoder mit dem gleichen Checkpoint eines Modells zu initialisieren, welches ursprünglich nur als Encoder trainiert wurde, beispielsweise mit BERT. Gemäß TL wird so im Kontext von NLP und ATS das Neuerlernen einer Sprache umgangen. Aufgrund der Verfügbarkeit von BERT wird in der Folge nur letztere Möglichkeit betrachtet. Die folgenden Ausführungen sind in Abbildung 4.1 visualisiert [Rothe et al., 2020, S. 2-3].

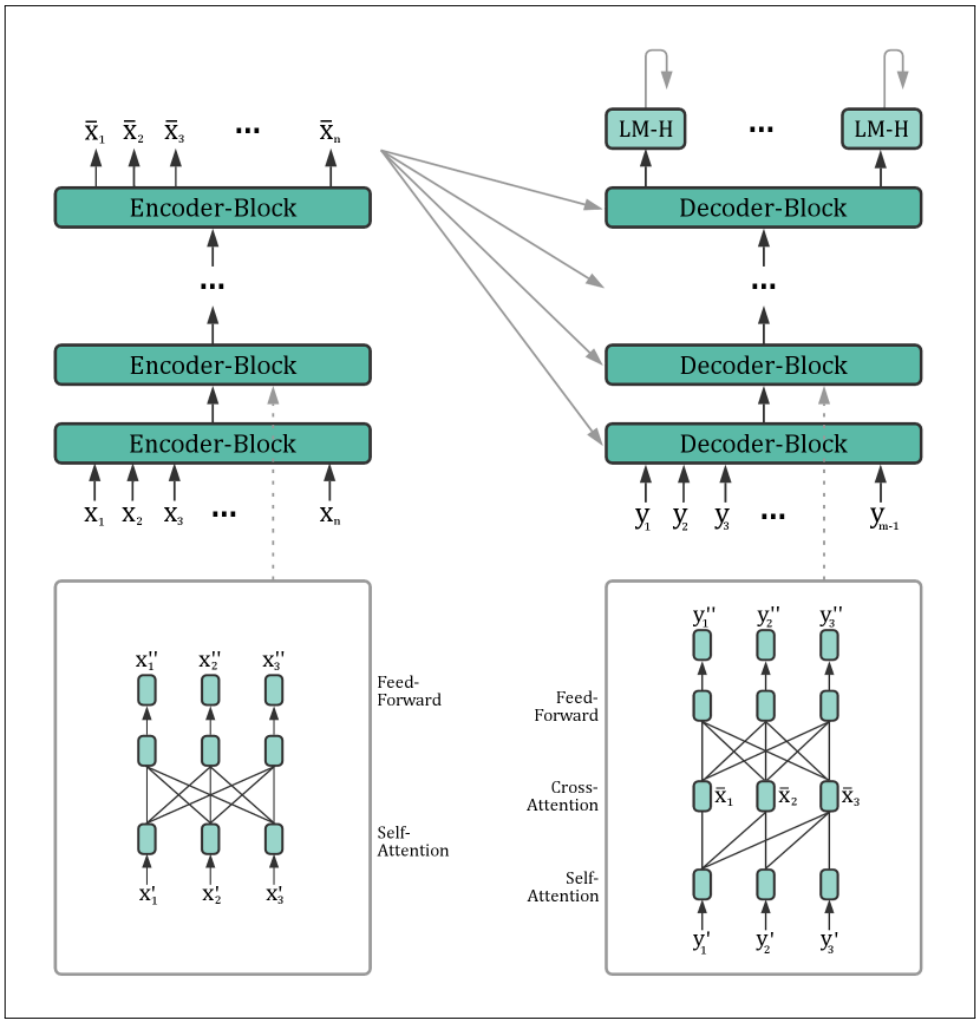

Abbildung 4.1: Sequence-to-Sequence-Transformer-Modell mit BERT [Von Platen, 2020].

Die Gewichte der bidirektionalen Self-Attention-Schichten und der Feed-Forward-Schichten aller Encoder-Blöcke werden mit den vortrainierten Gewichten von BERT initialisiert. Dabei kann der Encoder schlichtweg als BERT in seiner Reinform verstanden werden. Der Decoder hingegen bedarf mindestens der nachstehenden Anpassungen [Von Platen, 2020].

Zunächst werden sogenannte Cross-Attention-Schichten zwischen den Self-Attention-Schichten und den Feed-Forward-Schichten aller BERT-Blöcke eingeführt, um die kontextbasierten Sequenzen verarbeiten zu können. Die Gewichte der Cross-Attention-Schichten werden hierbei zufällig initialisiert [Von Platen, 2020].

Zudem werden die bidirektionalen Self-Attention-Schichten zu unidirektionalen Self-Attention-Schichten transformiert, um der autoregressiven Funktionsweise eines Decoders gerecht zu werden. Bei der autoregressiven NLG wird angenommen, dass die Wahrscheinlichkeitsverteilung einer Sequenz in das Produkt der bedingten nächsten Wortverteilungen zerlegt werden kann [Von Platen, 2020]. Beide Attention-Schichten basieren auf den gleichen Projektionen aus Key, Query und Value, weshalb die Gewichte dieser Schichten weiterhin mit den Werten von BERT initialisiert werden können. Die unidirektionalen Self-Attention-Schichten berücksichtigen nun nur noch vorangegangene Token, nicht mehr auch die nachstehenden Token. Dies führt zu veränderten Ausgabevektoren im Vergleich zum ursprünglich BERT, obwohl sie die gleichen Gewichte teilen [Rothe et al., 2020, S. 2].

Zuletzt wird dem Decoder eine sogenannte Language-Model-Head-Schicht hinzugefügt, dessen Gewichte mit denen des gewählten Word Embeddings initialisiert werden. Hierbei handelt es sich erneut um BERT. Es wird deutlich, dass sich der Encoder und der Decoder viele Gewichte teilen können. Dies führt zu einer erheblichen Reduktion des Speicherbedarfs, während die Qualität anschließender NLP-Aufgaben nahezu unverändert bleibt [Rothe et al., 2020, S. 2].

Die Textinhalte der Datengrundlage bedürfen überdies keiner weitergehenden Vorverarbeitung im herkömmlichen Sinne. Diese ist bekanntermaßen sehr individuell und stark modellabhängig. Unter Verwendung der als sehr robust geltenden Transformer-Architekturen entfällt daher die sonst übliche Textbereinigung sowie die Textnormalisierung. Dies unterliegt der Annahme, dass Transformer-Architekturen potenziell aus jeder Eigenart ein relevantes Feature schaffen können, welches das spätere Ergebnis begünstigt. Von der zugeführten Interpunktion und den vielfältigen Wortformen wird sich indes erhofft, potenzielle Mehr- oder Uneindeutigkeiten zu minimieren. Das Fine-Tuning sollte darüber hinaus stets unter gleichen Bedingungen wie das initiale Training stattfinden. Gleichzeitig sinkt hierdurch der vorverarbeitende Aufwand und damit auch etwaige Wartezeiten bei der praktischen Anwendung bereits trainierter Modelle in Echtzeit. Dennoch ist es möglich, bestimmte Vorverarbeitungsschritte a posteriori zu implementieren. Die Auswirkungen auf das Modell und die entsprechenden Ergebnisse würden somit zugleich messbar.

In der sonstigen technischen Vorbereitung ist weiterhin ein Tokenizer zu definieren. Dieser entstammt ebenfalls BERT und berücksichtigt Groß- und Kleinschreibung. BERT kann Sequenzen bis zu einer maximalen Länge von 512 Token verarbeiten. Dies unterschreitet zwar die durchschnittliche Textlänge der beschriebenen Korpora, kann jedoch unter der Annahme, dass wichtige Informationen zumeist am Anfang von Texten stehen, akzeptiert werden [Von Platen, 2020].

Von einer schlichten Erhöhung der maximalen Tokenlänge ist indes abzuraten, da hierbei ein quadratischer Anstieg der Rechenzeit und des Speicherbedarfs zu erwarten ist. Zudem wurde BERT ausschließlich auf Texten mit einer maximalen Tokenlänge von 512 trainiert. Ein denkbarer Lösungsansatz, welcher an dieser Stelle nur genannt sei, ist der sogenannte Sliding-Window-Approach. Hierbei besteht jedoch die Gefahr, dass langfristige Abhängigkeiten verloren gehen. Zuvor sind außerdem entsprechende Testläufe durchzuführen. Weiterhin existieren Ansätze wie etwa Longformer oder auch Big Bird, welche die Verarbeitung langer Sequenzen verfolgen [Zaheer et al., 2021]. Diese versuchen zugleich, lineare Komplexität zu erreichen, beispielsweise mithilfe lokaler Attention-Mechanismen, die wiederum mit globaler Attention verknüpft sind [Beltagy et al., 2020]. Ein eher anwendungsbezogener Workaround besteht hingegen darin, den jeweils eingehenden Rohtext alle 512 Token zu unterteilen, die Subtexte einzeln zusammenzufassen und die Zusammenfassungen zu konkatenieren. Dies betrifft jedoch nicht das eigentliche Training [Ding et al., 2020, S. 2].

Texte werden also zusammenfassend in den nachfolgenden Schritten jeweils nach 512 Token abgeschnitten. Die maximale Tokenlänge der entstehenden Zusammenfassungen wird auf 128 limitiert. Anpassungen, welche etwa im Laufe der experimentgetriebenen Entwicklung und Optimierung an Potenzial gewinnen, werden an den entsprechenden Stellen ergründet und evaluiert.

5 Datengrundlage

Um die Ziele dieser Arbeit zu erreichen, ist die Entwicklung theoretisch analysierter Architekturen zur ATS und zur sprachtechnischen Adaption erforderlich. Hierfür ist eine geeignete Datengrundlage bereitzustellen, welche insbesondere Qualität, aber auch Vergleichbarkeit der entsprechenden Modelle ermöglicht. Fortan wird die Datengrundlage als Korpus K bezeichnet, wobei dieser Korpus aus verschiedenen Datensätzen d_i besteht, also

$$K = \begin{pmatrix} d_1 \\ \vdots \\ d_n \end{pmatrix}$$

für $i = 1, ..., n$ mit möglichst großem n hinsichtlich hoher Qualität. Die Datensätze, welche den gesuchten Korpus bilden, müssen dabei bestimmten Anforderungen genügen. Ihnen wird insbesondere eine paarweise Natur abverlangt. Für $d_i \in K$ und $i = 1, ..., n$ gilt also: $d_i = \{t_i, s_i\}$. Neben dem ursprünglichen Text t_i ist hier eine Zusammenfassung s_i gefordert, welche als Referenz für die modellseitig zu generierende Zusammenfassung dient. Nur so ist die Qualität messbar und der Lernfortschritt realisierbar. Aufgrund der explorativen Natur dieser Arbeit werden sowohl englischsprachige als auch deutschsprachige Datensätze benötigt, wobei deren zugrundeliegende Domäne zunächst nicht von hoher Relevanz ist. Die Länge der Texte und der Zusammenfassungen haben einen hohen Einfluss darauf, wie das trainierte Modell die eigenen Zusammenfassungen generieren wird. Zwar wird hierfür keine Mindestlänge definiert, dennoch seien folgende Richtwerte gegeben: Texte t_i sollten aus mindestens 200 Wörtern bestehen. Zusammenfassungen s_i hingegen sollten einige Sätze vorweisen können. Alle Texte und Zusammenfassungen sollten darüber hinaus zwischen Klein- und Großschreibung unterscheiden.

Unter Berücksichtigung obiger Anforderungen werden nun vier Korpora ausgewählt. Diese werden wie folgt deklariert und nachfolgend beschrieben: K_a, K_b, K_c, K_d. Der Korpus K_a dient als initialer Trainingskorpus und besteht aus etwa 300.000 englischsprachigen Datensätzen. Er wurde von TensorFlow verarbeitet und veröffentlicht, entstammt allerdings ursprünglich der CNN und der DailyMail [TensorFlow, 2021]. Aufgrund der nach-

richtenorientierten Domäne ist von stark variierenden Textinhalten auszugehen. Dies verspricht zunächst einen hohen generalisierenden Effekt, wobei individuelle Zieldomänen womöglich andere Eigenarten aufweisen und mitunter eine andere Beschaffenheit des Korpus erfordern. Dies ist allerdings nicht Teil dieser Arbeit und gilt lediglich als sensibilisierende Anmerkung. Die Eignung des Korpus wird insbesondere durch die weitreichende Nutzung in der Wissenschaft bestärkt, denn SOTA-Modelle werden oftmals auf diesem Korpus verglichen. Texte dieses Korpus bestehen durchschnittlich aus etwa 850 Wörtern, Zusammenfassungen hingegen aus etwa 60 Wörtern. Dies spricht für einen hohen Abstraktionsgrad und damit eine hohe Verdichtung [Rothe et al., 2020, S. 6].

Die anderen drei Korpora dienen dem weitergehenden Training und bestehen demzufolge aus deutschsprachigen Datensätzen. Der Korpus K_b wurde 2019 im Kontext der Swiss Text Analytics Conference als Grundlage eines Wettbewerbes publiziert und umfasst 100.000 Datensätze [Cieliebak, 2019]. Die Textinhalte entstammen der deutschsprachigen Wikipedia, weshalb auch hier von einer vielfältigen Domäne auszugehen ist. Der Korpus K_c wurde durch einen in Python selbst entwickelten Crawler generiert. In einer Zeitspanne von sechs Monaten wurden mehr als 50.000 Nachrichtenartikel automatisiert kollektiert [Bird et al., 2009, S. 79, 83, 416]. Nach Sichtung der verfügbaren Daten können Artikel der ZEIT ONLINE als geeignet bewertet werden. Demnach sind etwa 15.000 Datensätze nutzbar. Der Korpus K_d nennt sich MLSUM und steht als multilingualer Korpus für das Training der ATS zur Verfügung. Die darin enthaltenen über 200.000 deutschsprachigen Datensätzen entstammen erneut vornehmlich einer nachrichtenorientierten Domäne [Scialom et al., 2020].

Texte der deutschsprachigen Korpora bestehen durchschnittlich aus etwa 4.000 Wörtern, Zusammenfassungen hingegen aus etwa 250 Wörtern. Üblicherweise existiert beim anvisierten Training die Gefahr der sogenannten Exploitation. Diese Gefahr meint im Kontext der ATS konkret, dass das zugrundeliegende Modell die Struktur der Artikel anstatt der Inhalte der Artikel lernt. Grund für diese Annahme ist der typische Aufbau von Wikipedia-Artikeln. Diese beinhalten zumeist bereits im ersten Absatz stark verdichtete Informationen, also eine Art Zusammenfassung. Dies macht eine hohe Anzahl an Trainingsdaten verschiedener Herkunft erforderlich. Die ersten Absätze der Wikipedia-Artikel werden beim Laden bereits ignoriert. Dennoch sollte zur Vorbeugung stets eine Mischung aus den drei deutschsprachigen Korpora vorgenommen werden [Bird et al., 2009, S. 42].

Um insbesondere der Kritik an den präsentierten Metriken entgegenzuwirken und belastbare Aussagen treffen zu können, wird nun ein englisch- und ein deutschsprachiger Korpus eingeführt, welche der qualitativen Analyse maschinell generierter Zusammenfassungen dienen (siehe Anhang A und B). Dabei ist strukturell und inhaltlich von korpusübergreifender Gleichheit auszugehen, während die beinhalteten Texte korpusintern möglichst unterschiedlich sind. Neben Berichten und Definitionen ist beispielsweise auch ein Rechtsurteil und eine Gebrauchsanweisung enthalten. Die Texte können hierbei der Allgemeinsprache, der Fachsprache und der Alltagssprache zugeordnet werden. Die Texte entstammen bundesweiten Informations- und Nachrichtenkanälen.

6 Experimente

Unter Kenntnis der Grundlagen des DL, des NLP und der ATS wird nun ein Ablauf verschiedener Experimente konzipiert. Zuvor wird die entsprechende Entwicklungsumgebung offengelegt.

6.1 Entwicklungsumgebung

Die Entwicklung und die Durchführung aller Experimente geschieht in Python. Dies ist eine Programmiersprache, welche sich insbesondere für ML- und DL-Zwecke eignet. Dabei werden Trainingsprozesse durch Compute Unified Device Architecture (CUDA) unterstützt, wenn entsprechende Voraussetzungen erfüllt sind. CUDA ist eine von NVIDIA entwickelte Technik, welche es ermöglicht, bestimmte Operationen mithilfe der GPU zu beschleunigen [NVIDIA, 2021]. Zudem ist es in dieser Umgebung möglich, vortrainierte Modelle wie BERT, XLM-R und BART zu laden und Architekturen weitergehend gemäß der bereits bekannten Konfiguration zu präparieren, darunter beispielsweise die beschriebene Encoder-Decoder-Architektur. Dies wird durch die Bibliothek PyTorch und das US-Unternehmen HuggingFace, welches den Code als Open Source bereitstellt, ermöglicht. HuggingFace stellt zudem verschiedene Klassen zum Trainieren von Sequence-to-Sequence-Modellen bereit [HuggingFace, 2021]. Darüber hinaus erfolgen alle Experimente dieser Arbeit über einen legitimierten Zugang auf dem Hochleitungsrechner der TU Dresden, namentlich Taurus, um das Potenzial der verfügbaren Umgebung mithilfe einer leistungsstarken GPU (NVIDIA V100) vollends auszuschöpfen [ZIH, 2021]. Der Quellcode ist dem Anhang zu entnehmen.

6.2 Reproduktion auf englischen Daten

Zunächst erfolgt die Reproduktion des SOTA-Benchmark auf Grundlage der konzipierten Architektur, um eine Baseline zu setzen, an welcher sich in nachfolgenden Experimenten verglichen und gemessen werden kann. Daher ist es unabdingbar, je ein erstes Modell unter Nutzung von BERT, XLM-R und BART auf dem englischsprachigen Korpus zu trainieren und zu evaluieren. Dies folgt der beschriebenen Architektur und der entsprechenden Konfiguration ohne Kompromisse.

6.3 Adaption auf deutschen Daten

Anschließend wird die Adaption auf die deutsche Sprache erprobt. Die bislang genutzte Architektur, welche bekanntermaßen Transformer integriert, kann auf Grundlage umfangreicher verschiedensprachiger ungelabelter Textdaten im Sinne des DL und TL multilingual vortrainiert werden, ohne architektonische Anpassungen vornehmen zu müssen. Hierbei werden sprachübergreifende verborgene Strukturen erlernt, um anschließend monolingual davon zu profitieren. Zudem wird dem Problem entgegengewirkt, dass sprachintern zu wenig Textdaten zur Verfügung stehen [Moberg, 2020]. Modelle, welche organisationsextern bereits multilingual vortrainiert und bereitgestellt wurden, sind beispielsweise die multilinguale Version von BERT, XLM-R und BART. Hinsichtlich einer anschließenden deutschsprachigen Nutzung in der ATS ist folglich ein entsprechendes sprachbezogenes Training erforderlich. Hierfür werden die drei oben genannten Modelle mit einem Mix aus allen deutschsprachigen Daten erneut trainiert und evaluiert.

6.4 Adaption auf multilingualen Daten

Basierend auf der Annahme, multilinguale Modelle würden von verborgenen Strukturen anderer Sprachen profitieren, werden die drei bewährten Modelle nun jeweils mit einem Mix aus allen englisch- und deutschsprachigen Daten trainiert und auf Basis einer deutschsprachigen Evaluation mit den bisherigen Fortschritten verglichen.

6.5 Optimierung der Hyperparameter

Weitere Experimente, wie beispielsweise die Optimierung der Hyperparameter oder die Variation der jeweiligen Korpusanteile, werden im nächsten Kapitel nach der grundlegenden Evaluation der hier konzipierten Experimente abgeleitet und durchgeführt.

7 Evaluation

In diesem Kapitel werden die zuvor konzipierten Experimente umfangreich evaluiert. Dies geschieht einerseits auf Basis der ausgewählten Metriken, andererseits in Form einer qualitativen Analyse der aus den Anhängen A und B generierten Zusammenfassungen.

7.1 Automatische Auswertung

Text...

7.2 Qualitative Analyse

Text...

TODO's:
+ Empfehlung pro Experimente/ pro Sprache/ pro Modell, bspw. bzgl. Domäneneignung, Kritik
BERT-EN:
Das entsprechende Training verlief komplikationslos und mit der erwarteten Fehlerabnahme. Letztlich sind die folgenden ROUGE-Scores zu verzeichnen: R-Recall: 15.78, R-Precision: 10.25, R-Measure: 12.11. Diese stimmen unter Akzeptanz womöglich unbekannter Nebenbedingungen mit den umworbenen ROUGE-Scores des SOTA überein.

Zudem wird der in Anhang A befindliche englischsprachige Text mithilfe des trainierten Modells zusammengefasst. Der Text unterscheidet sich strukturell von den eingegangenen Trainingsdaten und weist etwas über 1.000 Token auf. Weiterhin entstammt er einer dem Modell unbekannten Domäne. Fachlich ist er zugleich nicht zu spezifisch. Er handelt von den Anschlägen des 11. Septembers 2001 in New York. Die entstehende Zusammenfassung ist nachfolgend einzusehen. Es ist aus menschlich subjektiver Sicht erkennbar, dass nicht nur die verdichteten Informationen, sondern auch die Grammatik

und die Orthographie weitestgehend, aber nicht vollumfänglich, korrekt sind.

Firefighters, police officers, search-and-rescue dogs headed to Ground Zero to search for survivors. They didn't know how many people were trapped alive in the wreckage. They were able to search through the unstable piles of rubble for air pockets. By September 11th, workers had rescued all the people trapped at the site.

BERT-DE:

In einem initialen Experiment wird die multilinguale Version von BERT mit einem Mix aus allen verfügbaren deutschsprachigen Daten, also letztlich monolingual, trainiert. Hierbei entstehen die folgenden ROUGE-Scores: R-Recall: 15.93, R-Precision: 10.52, R-Measure: 11.97. Dies entspricht dem SOTA, wenn man anhand der vorliegenden Werte urteilt. Nachfolgend sei die fehlerbehaftete Zusammenfassung des in Anhang B einzusehenden deutschsprachigen Textes gezeigt. Dieser gleicht strukturell sowie inhaltlich weitestgehend dem bereits beschriebenen Text in Anhang A.

Der 11. September 2001 war der größte Unfall in der Geschichte der USA. Doch die Bilder zeigen, wie groß die Welt ist, und wie groß der Druck auf die Menschen ist. 500 Feuerwehrleute und Polizeibeamte starben. New York steht unter Schock und vor einem Desaster für die USA. Ein Gedenkpavillon erinnert an die Aufräumarbeiten auf dem World Trade Center.

Die inhaltliche Insuffizienz ist für den menschlichen Leser auch ohne Kenntnis über den Originaltext unschwer erkennbar. Dies ist aus praktischer Sicht zwar nachteilig, ist aber auch im englischsprachigen SOTA existent, wenn auch subjektiv weniger gravierend. Die Grammatik und die Orthographie hingegen scheinen fehlerfrei zu sein.

Testweise wird nun die multilinguale Version von BERT durch eine eigens für die deutsche Sprache vortrainierte Version ausgetauscht. Hierbei handelt es sich um den sogenannten German BERT (GBERT), welcher von Deepset AI, einem globalen Anbieter von Open-Source-Lösungen für NLP, vortrainiert und bereitgestellt wird. An der deutschsprachigen Datengrundlage wird zunächst nichts verändert. Hierbei entstehen die folgenden ROUGE-Scores: R-Recall: 17.83, R-Precision: 10.79, R-Measure: 12.72. Nachfolgend ist erneut eine Zusammenfassung des Textes aus Anhang B zu sehen.

Der 11. September 2001 war der erste Terroranschlag in der Geschichte der USA. Die Bilder der Katastrophe zeigen, wie sehr die USA sich auf die Welt vorbereiten - und wie sich die Welt verändert hat. Das World Trade Center ist ein Symbol für die Zerstörung der Welt. Es ist ein Ort der Trauer und der Trauer. Das Museum zeigt die Ereignisse des 11. Septembers.

Obwohl die ROUGE-Scores den SOTA übersteigen, ist die exemplarische Zusammenfassung inhaltlich noch immer sehr fehlerbehaftet. Daher werden in einem folgenden Experiment die weitergehenden Potenziale der multilingualen Version von BERT erprobt. Hierfür werden nun im Trainingsprozess alle verfügbaren Daten, bestehend aus englisch- und deutschsprachigen Texten, unter sonst gleichen Bedingungen berücksichtigt. Dabei entstehen die folgenden ROUGE-Scores: R-Recall: 16.76, R-Precision: 11.18, R-Measure: 12.68. Es ist festzuhalten, dass das Training auf einer multilingualen Datenbasis sowohl wertmäßig als auch subjektiv qualitativ bessere Ergebnisse als das Training auf einer monolingualen Datenbasis hervorbringt. Nachfolgend wird dies anhand der Zusammenfassung des Anhang B deutlich.

Der 11. September 2001 war der größte Unfall in der Geschichte der Welt. Doch die Bilder, die an diesem Mittwoch in New York passierten, sind fassungslos. Fast 400 Feuerwehrleute und Polizeibeamte verloren bei den Rettungsarbeiten ihr Leben. Der traurige Tag ging fortan als 9/11 in die Geschichte ein.

Trotz allen bisherigen Fortschritten sind weiterhin inhaltliche Mängel in den generierten Zusammenfassungen zu erkennen. Mit dem Wissen, dass das Training auf einer multilingualen Datenbasis bessere Ergebnisse hervorbringt, wird nun BERT durch XLM-R ausgetauscht und unter sonst gleichen Bedingungen trainiert. Das Training basiert folglich ebenfalls auf allen verfügbaren Daten, strebt die vermuteten Potenziale von XLM-R an und bringt die folgenden ROUGE-Scores hervor: R-Recall: 0.00, R-Precision: 0.00, R-Measure: 0.00. Nachfolgend ist die Zusammenfassung des Textes aus Anhang B zu sehen.

...

Trotz der monolingualen Fortschritte wird GBERT zunächst durch XLM-R und anschließend durch BART ersetzt, um sich weiterhin geeigneten Ergebnissen zu nähern und multilinguale Modelle zu erproben. Der Tokenizer wird ebenfalls entsprechend ausge-

tauscht. Es entstehen die folgenden ROUGE-Scores: R-Recall: 23.02, R-Precision: 13.95, R-Measure: 16.09. Obgleich XLM-R bewiesenermaßen verschiedene NLP-Aufgaben begünstigt, kann sie in der vorliegenden Konfiguration keine Verbesserung der ATS erzielen. Dies basiert nicht nur auf den verringerten ROUGE-Scores, sondern auch auf der nachfolgenden exemplarischen Zusammenfassung von Anhang B, welche inhaltlich subjektiv als mangelhaft eingestuft werden kann.

Der 11. Flugtag der Weltturm-Weltmeisterschaft 2001 fand am 11. September 2001 in New York City statt und war das erste Mal in der Geschichte des World Trade Centers. Das National September 11 Memorial and Museum in Manhattan ist ein historischer Gedenkpavillon in Manhattan. Es befindet sich in der Nähe des World Trade Centers in Manhattan.

In der Konsequenz wird GBERT für die weiteren Experimente genutzt. Hierbei werden insbesondere die Anteile der jeweiligen Korpora an den Trainings- und Testdaten variiert, um vornehmlich dem Lernen der Textstruktur entgegenzuwirken. Die Konfigurationen sowie entsprechende Ergebnisse sind der Tabelle ? zu entnehmen.

- Training auf Wiki, Evaluation auf News -¿ Sehr schlecht, strukturerlernend - Training auf News mit ein bisschen Wiki, Evaluation mit New -¿ Positiver Effekt, aber zu wenig Daten - Training auf News und 50 Prozent von Wiki, Evaluation mit News -¿ ? - Training auf übersetztem CNN/ Dailymail, Evaluation mit News -¿ ?

Modell	**R-Recall**	**R-Precision**	**R-Measure**
BERT	15.78	10.25	12.11
XLM-R	0	0	0
BART	0	0	0

Tabelle 7.1: SOTA-Reproduktion im Modellvergleich.

8 Zusammenfassung

9 Diskussion und Ausblick

Literaturverzeichnis

[AI-United, 2019] AI-United: LSTM als neuronales Netzwerk mit langem Kurzzeitgedächtnis, in: http://www.ai-united.de/lstm-als-neuronales-netzwerk-mit-langem-kurzzeitgedaechtnis/, Aufruf am 01.03.2021.

[Arbel, 2018] Arbel, Nir: How LSTM Networks Solve the Problem of Vanishing Gradients, in: https://medium.datadriveninvestor.com/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577, Aufruf am 01.03.2021.

[Beltagy et al., 2020] Beltagy, Iz & Peters, Matthew & Cohan, Arman: Longformer as the Long-Document Transformer, Allen Institute for Artificial Intelligence, Seattle, 2020.

[Bird et al., 2009] Bird, Steven & Klein, Ewan & Loper, Edward: Natural Language Processing with Python, Verlag O'Reilly, Sebastopol, Vereinigte Staaten, 2009.

[Brownlee, 2019] Brownlee, Jason: A Gentle Introduction to the Bag-of-Words Model, in: https://machinelearningmastery.com/gentle-introduction-bag-words-model/, Aufruf am 01.03.2021.

[Cieliebak, 2019] Cieliebak, Mark: German Text Summarization Challenge, Swiss Text Analytics Conference, Winterthur, 2019.

[Conneau et al., 2020] Conneau, Alexis & Khandelwal, Kartikay & Goyal, Naman & Chaudhary, Vishrav & Wenzek, Guillaume & Guzman, Francisco & Grave, Edouard & Ott, Myle & Zettlemoyer, Luke & Stoyanov, Veselin: Unsupervised Cross-Lingual Representation Learning at Scale, Facebook AI, 2020.

[CS231N, O. J.] Convolutional Neural Networks for Visual Recognition: Nesterov Momentum, in: https://cs231n.github.io/neural-networks-3/, Aufruf am 01.03.2021.

[Culurciello, 2018] Culurciello, Eugenio: The Fall of RNN/ LSTM, in: https://towardsdatascience.com/the-fall-of-rnn-lstm-2d1594c74ce0, Aufruf am 01.03.2021.

[Devlin et al., 2019] Devlin, Jacob & Chang, Ming-Wei & Lee, Kenton & Toutanova, Kristina: Pre-training of Deep Bidirectional Transformers for Language Understanding, Google AI Language, 2019.

[Ding et al., 2020] Ding, Ming & Zhou, Chang & Yang, Hongxia & Tang, Jie: Applying BERT to Long Texts, in: https://proceedings.neurips.cc/paper/2020/file/96671501524948bc3937b4b30d0e57b9-Paper.pdf, Aufruf am 01.03.2021.

[Edpresso, O. J.] Edpresso: Overfitting and Underfitting, in: https://www.educative.io/edpresso/overfitting-and-underfitting, Aufruf am 01.03.2021.

[Gambhir et al., 2016] Gambhir, Mahak & Gupta, Vishal: Recent Automatic Text Summarization Techniques, University of Panjab in Chandigargh, 2016.

[Goncalves, 2020] Goncalves, Luis: Automatic Text Summarization with Machine Learning, in: https://medium.com/luisfredgs/automatic-text-summarization-with-machine-learning-an-overview-68ded5717a25, Aufruf am 01.03.2021.

[HuggingFace, 2021] HuggingFace: The AI community building the future, in: https://huggingface.co/, Aufruf am 01.03.2021.

[Huilgol, 2020] Huilgol, Purva: Quick Introduction to Bag-of-Words (BoW) and TF-IDF for Creating Features from Text, in: https://www.analyticsvidhya.com/blog/2020/02/quick-introduction-bag-of-words-bow-tf-idf/, Aufruf am 01.03.2021.

[Irene, 2018] Irene: ELMo in Practive, in: https://ireneli.eu/2018/12/17/elmo-in-practice/, Aufruf am 01.03.2021.

[Karani, 2018] Karani, Dhruvil: Introduction to Word Embedding and Word2Vec, in: https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa, Aufruf am 01.03.2021.

[Karim, 2019] Karim, Raimi: Self-Attention, in: https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a, Aufruf am 01.03.2021.

[Khanna, 2019] Khanna, Sachin: Machine Learning vs. Deep Learning, Department of Computer Science Engineering in India, 2019.

[Kiani, 2017] Kiani, Farzad: Automatic Text Summarization, University of Arel in Istanbul, 2017.

[Kingma et al., 2017] Kingma, Diederik & Ba, Jimmy: Adam - A Method for Stochastic Optimization, University of Amsterdam and Toronto, 2017.

[Kriesel, 2005] Kriesel, David: Ein kleiner Überblick über neuronale Netze, in: http://www.dkriesel.com/science/neural_networks, Aufruf am 01.03.2021.

[Lemberger, 2020] Lemberger, Pirmin: Deep Learning Models for Automatic Summarization, in: https://towardsdatascience.com/deep-learning-models-for-automatic-summarization-4c2b89f2a9ea, Aufruf am 01.03.2021.

[Lewis et al., 2019] Lewis, Mike & Liu, Yinhan & Goyal, Naman & Ghazvininejad, Marjan & Mohamed, Abdelrahman & Levy, Omer & Stoyanov, Ves & Zettlemoyer, Luke: BART as Denoising Sequence-to-Sequence Pre-Training for Natural Language Generation, Translation and Comprehension, Facebook AI, 2019.

[Lin, 2004] Lin, Chin-Yew: ROUGE as a Package for Automatic Evaluation of Summaries, Information Sciences Institute, Southern California, 2004.

[Luber et al., 2018] Luber, Stefan & Litzel, Nico: Long-Short-Term-Memory, in: https://www.bigdata-insider.de/was-ist-ein-long-short-term-memory-a-774848/, Aufruf am 01.03.2021.

[Manning et al., 2008] Manning, Christopher & Raghavan, Prabhakar & Schütze, Heinrich: Introduction to Information Retrieval, Cambridge University Press, 2008.

[McCullum, 2020] McCullum, Nick: Deep Learning Neural Networks Explained, in: https://www.freecodecamp.org/news/deep-learning-neural-networks-explained-in-plain-english/, Aufruf am 01.03.2021.

[Moberg, 2020] Moberg, John: A Deep Dive into Multilingual NLP Models, in: https://peltarion.com/blog/data-science/a-deep-dive-into-multilingual-nlp-models, Aufruf am 01.03.2021.

[Nallapati et al., 2016] Nallapati, Ramesh & Zhou, Bowen & Dos Santos, Cicero & Gulcehre, Caglar & Xiang, Bing: Abstractive Text Summarization using Sequence-to-Sequence RNNs, Conference on Computational Natural Language Learning, 2016.

[Nitsche, 2019] Nitsche, Matthias: Towards German Abstractive Text Summarization using Deep Learning, HAW Hamburg, 2019.

[NLTK, 2020] NLTK: Stem-Package, in: https://www.nltk.org/api/nltk.stem.html, Aufruf am 01.03.2021.

[NVIDIA, 2021] NVIDIA: CUDA Toolkit, in: https://developer.nvidia.com/cuda-toolkit, Aufruf am 01.03.2021.

[Papineni et al., 2002] Papineni, Kishore & Roukos, Salim & Ward, Todd & Zhu, Wei-Jing: BLEU as a Method for Automatic Evaluation of Machine Translation, Association for Computational Linguistics, Philadelphia, 2002.

[Paulus et al., 2017] Paulus, Romain & Xiong, Caiming & Socher, Richard: A Deep Reinforced Model for Abstractive Summarization, in: https://arxiv.org/pdf/1705.04304v3.pdf, Aufruf am 01.03.2021.

[Penninglon et al., 2014] Penninglon, Jeffrey & Socher, Richard & Manning, Christopher: Global Vectors for Word Representation, Stanford University, 2014.

[Peters et al., 2018] Peters, Matthew & Neumann, Mark & Iyyer, Mohit & Gardner, Matt & Clark, Christopher & Lee, Kenton & Zettlemoyer, Luke: Deep Contextualized Word Representations, Allen Institute for Artificial Intelligence, Washington, 2018.

[Radford et al., 2019] Radford, Alec & Wu, Jeff & Child, Rewon & Luan, David & Amodei, Dario & Sutskever, Ilya: Language Models are Unsupervised Multitask Learners, in: https://openai.com/blog/better-language-models/, Aufruf am 01.03.2021.

[Raffel et al., 2020] Raffel, Colin & Shazeer, Noam & Roberts, Adam & Lee, Katherine & Narang, Sharan & Matena, Michael & Zhou, Yanqi & Li, Wei & Lio, Peter: Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer, Google Research, 2020.

[Raschka et al., 2019] Raschka, Sebastian & Mirjalili, Vahid: Machine Learning and Deep Learning with Python, Verlag Packt, Birmingham, Vereinigtes Königreich, 2019.

[Rothe et al., 2020] Rothe, Sascha & Narayan, Shashi & Severyn, Aliaksei: Leveraging Pre-Trained Checkpoints for Sequence Generation Tasks, Google Research, 2020.

[Scialom et al., 2020] Scialom, Thomas & Dray, Paul-Alexis & Lamprier, Sylvain & Piwowarski, Benjamin & Staiano, Jacopo: MLSUM as the Multilingual Summarization Corpus, Sorbonne Université, Paris, 2020.

[SpaCy, 2021] SpaCy: German Models, in: `https://spacy.io/models/de`, Aufruf am 01.03.2021.

[TensorFlow, 2021] TensorFlow: Datasets - CNN-DailyMail, in: `https://www.tensorflow.org/datasets/catalog/cnn_dailymail`, Aufruf am 01.03.2021.

[Thaker, 2019] Thaker, Madhav: Comparing Text Summarization Techniques, in: `https://towardsdatascience.com/comparing-text-summarization-techniques-d1e2e465584e`, Aufruf am 01.03.2021.

[Vaswani et al., 2017] Vaswani, Ashish & Shazeer, Noam & Parmar, Niki & Uszkoreit, Jakob & Jones, Llion & Gomez, Aidan & Kaiser, Lukasz & Polosukhin, Illia: Attention Is All You Need, Google Research, 2017.

[Von Platen, 2020] Von Platen, Patrick: Leveraging Pre-trained Language Model Checkpoints for Encoder-Decoder Models, in: `https://huggingface.co/blog/warm-starting-encoder-decoder`, Aufruf am 01.03.2021.

[Yang et al., 2019] Yang, Liu & Lapata, Mirella: Text Summarization with Pretrained Encoders, Institute for Language, Cognition and Computation in Edinburgh, 2019.

[Yang et al., 2020] Yang, Li & Shami, Abdallah: On Hyperparameter Optimization of Machine Learning Algorithms, University of Western Ontario, 2020.

[Zaheer et al., 2021] Zaheer, Manzil & Guruganesh, Guru & Dubey, Avinava & Ainslie, Joshua & Alberti, Chris & Ontanon, Santiago & Pham, Philip & Ravula, Anirudh & Wang, Qifan & Yang, Li & Ahmed, Amr: Bid Bird as a Transformer for Longer Sequences, Google Research, 2020.

[Zhang et al., 2020] Zhang, Aston & Lipton, Zachary & Li, Mu & Smola, Alexander: Dive into Deep Learning, in: `https://d2l.ai/`, Aufruf am 01.03.2021.

[ZIH, 2021] Zentrum für Informationsdienste und Hochleitungsrechnen der TU Dresden: Hochleitungsrechnen, in: `https://tu-dresden.de/zih/hochleistungsrechnen`, Aufruf am 01.03.2021.

Thesen

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet.

Selbstständigkeitserklärung

Hiermit erkläre ich, Daniel Vogel, die vorliegende Masterarbeit selbstständig und nur unter Verwendung der von mir angegebenen Literatur verfasst zu haben. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegen.

Dresden, den ???. Juli 2021

Daniel Vogel

Quellcode

Konfigurationsdatei

```
language = "german" # english, german, multilingual
model_name = "bert-base-multilingual-cased"
tokenizer_name = "bert-base-multilingual-cased"
batch_size = 16

ratio_corpus_wiki = 1.00
ratio_corpus_news = 1.00
ratio_corpus_mlsum = 1.00
ratio_corpus_eng = 1.00

path_output = "/scratch/ws/1/davo557d-ws_project/"
path_checkpoint = "/scratch/ws/1/davo557d-ws_project/checkpoint-100000"

text_english = "..."
text_german = "..."
```

Quelltext 9.1: Konfigurationsdatei

Hilfsmethoden

```
# Imports
import gc
import csv
import torch
import psutil
import datasets
import transformers
import pandas as pd
import tf2tf_tud_gpu_config as config

from datasets import ClassLabel


# Methods
def load_data(language, ratio_corpus_wiki=0.0, ratio_corpus_news=0.0, ratio_corpus_mlsum
    =0.0, ratio_corpus_eng=0.0):
    if str(language) == "english":
        return load_english_data()

    else:
        # CORPUS: WIKI
        data_txt, data_ref = [], []

        with open("./data_train.csv", "r", encoding="utf-8") as f:
            reader = csv.reader(f, delimiter=",", quoting=csv.QUOTE_ALL)
            next(reader, None)

            for row in reader:
                data_txt.append(row[0])
                data_ref.append(row[1])

        ds_wiki = datasets.arrow_dataset.Dataset.from_pandas(
            pd.DataFrame(
                list(zip(data_txt, data_ref)),
                columns=["text", "summary"]
            )
        )

        # CORPUS: NEWS
        df_news = pd.read_excel("./data_train_test.xlsx", engine="openpyxl")
        df_news = df_news[["article", "highlights"]]
        df_news.columns = ["text", "summary"]
        df_news = df_news[~df_news["summary"].str.contains("ZEIT")]
```

```
43          df_news = df_news.dropna()
44          ds_news = datasets.arrow_dataset.Dataset.from_pandas(df_news)
45          ds_news = ds_news.remove_columns("__index_level_0__")
46
47          # CORPUS: MLSUM
48          ds_mlsum = datasets.load_dataset("mlsum", "de", split="train")
49          ds_mlsum = ds_mlsum.remove_columns(["topic", "url", "title", "date"])
50
51          text_corpus_mlsum = []
52          summary_corpus_mlsum = []
53
54          for entry in ds_mlsum:
55              text = entry["text"]
56              summary = entry["summary"]
57
58              if summary in text:
59                  text = text[len(summary) + 1:len(text)]
60
61              text_corpus_mlsum.append(text)
62              summary_corpus_mlsum.append(summary)
63
64          ds_mlsum = datasets.arrow_dataset.Dataset.from_pandas(
65              pd.DataFrame(
66                  list(zip(text_corpus_mlsum, summary_corpus_mlsum)),
67                  columns=["text", "summary"]
68              )
69          )
70
71          # ACTION: CONCAT
72          german_data = datasets.concatenate_datasets([
73              ds_wiki.select(
74                  range(0, int(len(ds_wiki) * ratio_corpus_wiki))),
75              ds_news.select(
76                  range(0, int(len(ds_news) * ratio_corpus_news))),
77              ds_mlsum.select(
78                  range(0, int(len(ds_mlsum) * ratio_corpus_mlsum)))
79          ])
80
81          if str(language) == "multilingual":
82              english_data, _, _ = load_english_data()
83              english_data = english_data.remove_columns("id")
84
85              prepared_data = datasets.concatenate_datasets([
86                  german_data.shuffle(),
87                  english_data.select(
88                      range(0, int(len(english_data) * ratio_corpus_eng))
```

```
                    ).shuffle()
                ])

        else:
            prepared_data = german_data.shuffle()

        # ACTION: SPLIT
        train_size = int(len(prepared_data) * 0.900)
        valid_size = int(len(prepared_data) * 0.005)  # 0.025
        test_size = int(len(prepared_data) * 0.075)

        train_data = prepared_data.select(
            range(0, train_size))
        val_data = prepared_data.select(
            range(train_size, train_size + valid_size))
        test_data = prepared_data.select(
            range(train_size + valid_size, train_size + valid_size + test_size))

        del prepared_data

        return train_data.shuffle(), val_data.shuffle(), test_data.shuffle()


def load_english_data():
    train_data = datasets.load_dataset(
        "cnn_dailymail", "3.0.0", split="train",
         ignore_verifications=True)
    val_data = datasets.load_dataset(
        "cnn_dailymail", "3.0.0", split="validation[:10%]",
         ignore_verifications=True)
    test_data = datasets.load_dataset(
        "cnn_dailymail", "3.0.0", split="test[:5%]",
         ignore_verifications=True)

    train_data = train_data.rename_column("article", "text")
    train_data = train_data.rename_column("highlights", "summary")
    val_data = val_data.rename_column("article", "text")
    val_data = val_data.rename_column("highlights", "summary")
    test_data = test_data.rename_column("article", "text")
    test_data = test_data.rename_column("highlights", "summary")

    return train_data, val_data, test_data


def test_cuda():
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
135     torch.cuda.empty_cache()
136
137     print("Device:", device)
138     print("Version:", torch.__version__)
139
140
141 def explore_corpus(data):
142     df = pd.DataFrame(data)
143
144     text_list = []
145     summary_list = []
146
147     for index, row in df.iterrows():
148         text = row["text"]
149         summary = row["summary"]
150         text_list.append(len(text))
151         summary_list.append(len(summary))
152
153     df = pd.DataFrame(data[:1])
154
155     for column, typ in data.features.items():
156         if isinstance(typ, ClassLabel):
157             df[column] = df[column].transform(lambda i: typ.names[i])
158
159
160 def empty_cache():
161     gc.collect()
162     torch.cuda.empty_cache()
163     psutil.virtual_memory()
164
165     # print(torch.cuda.get_device_properties(0).total_memory)
166     # print(torch.cuda.memory_reserved(0))
167     # print(torch.cuda.memory_allocated(0))
168
169
170 def load_tokenizer_and_model(from_checkpoint=False):
171     tokenizer = transformers.AutoTokenizer.from_pretrained(
172         config.tokenizer_name, strip_accent=False  # add_prefix_space=True
173     )
174
175     if from_checkpoint:
176         if "mbart" in config.model_name:
177             tf2tf = transformers.AutoModelForSeq2SeqLM.from_pretrained(
178                 config.path_checkpoint
179             )
180
```

```
181             else:
182                 tf2tf = transformers.EncoderDecoderModel.from_pretrained(
183                     config.path_checkpoint
184                 )
185 
186         else:
187             if "mbart" in config.model_name:
188                 tf2tf = transformers.AutoModelForSeq2SeqLM.from_pretrained(
189                     config.model_name
190                 )
191 
192             else:
193                 tf2tf = transformers.EncoderDecoderModel.from_encoder_decoder_pretrained(
194                     config.model_name, config.model_name, tie_encoder_decoder=True
195                 )
196 
197         return tokenizer, tf2tf
198 
199 
200 def configure_model(tf2tf, tokenizer):
201     tf2tf.config.decoder_start_token_id = tokenizer.cls_token_id
202     tf2tf.config.bos_token_id = tokenizer.bos_token_id
203     tf2tf.config.eos_token_id = tokenizer.sep_token_id
204     tf2tf.config.pad_token_id = tokenizer.pad_token_id
205     # tf2tf.config.vocab_size = tf2tf.config.encoder.vocab_size
206 
207     tf2tf.config.max_length = 128
208     tf2tf.config.min_length = 56
209     tf2tf.config.no_repeat_ngram_size = 3
210     tf2tf.config.early_stopping = True
211     tf2tf.config.length_penalty = 2.0
212     tf2tf.config.num_beams = 2
213 
214     return tf2tf
```

Quelltext 9.2: Hilfsmethoden

Trainingscode

```
# Imports
import datasets
import transformers
import tf2tf_tud_gpu_config as config
import tf2tf_tud_gpu_helpers as helpers


# Main
tokenizer, tf2tf = helpers.load_tokenizer_and_model(from_checkpoint=False)

train_data, val_data, test_data = helpers.load_data(
    language=config.language,
    ratio_corpus_wiki=config.ratio_corpus_wiki,
    ratio_corpus_news=config.ratio_corpus_news,
    ratio_corpus_mlsum=config.ratio_corpus_mlsum,
    ratio_corpus_eng=config.ratio_corpus_eng
)

helpers.test_cuda()
helpers.explore_corpus(train_data)
helpers.empty_cache()
rouge = datasets.load_metric("rouge")

tf2tf = helpers.configure_model(tf2tf, tokenizer)
tf2tf.to("cuda")


def process_data_to_model_inputs(batch):
    encoder_max_length = 512
    decoder_max_length = 128

    inputs = tokenizer(batch["text"], padding="max_length",
                       truncation=True, max_length=encoder_max_length)

    outputs = tokenizer(batch["summary"], padding="max_length",
                        truncation=True, max_length=decoder_max_length)

    batch["input_ids"] = inputs.input_ids
    batch["attention_mask"] = inputs.attention_mask
    batch["decoder_input_ids"] = outputs.input_ids
    batch["decoder_attention_mask"] = outputs.attention_mask
    batch["labels"] = outputs.input_ids.copy()
```

```
43      batch["labels"] = [[-100 if token == tokenizer.pad_token_id else token for token in
            labels]
44                         for labels in batch["labels"]]
45
46      return batch
47
48
49  train_data = train_data.map(
50      process_data_to_model_inputs,
51      batched=True,
52      batch_size=config.batch_size,
53      remove_columns=["text", "summary"]
54  )
55
56  train_data.set_format(
57      type="torch",
58      columns=["input_ids",
59               "attention_mask",
60               "decoder_input_ids",
61               "decoder_attention_mask",
62               "labels"]
63  )
64
65  val_data = val_data.map(
66      process_data_to_model_inputs,
67      batched=True,
68      batch_size=config.batch_size,
69      remove_columns=["text", "summary"]
70  )
71
72  val_data.set_format(
73      type="torch",
74      columns=["input_ids",
75               "attention_mask",
76               "decoder_input_ids",
77               "decoder_attention_mask",
78               "labels"]
79  )
80
81
82  def compute_metrics(pred):
83      labels_ids = pred.label_ids
84      pred_ids = pred.predictions
85
86      pred_str = tokenizer.batch_decode(pred_ids, skip_special_tokens=True)
87      labels_ids[labels_ids == -100] = tokenizer.pad_token_id
```

```
88      label_str  = tokenizer.batch_decode(labels_ids,  skip_special_tokens =True)
89
90      rouge_output = rouge.compute(
91          predictions=pred_str,
92          references=label_str,
93          rouge_types=["rouge2"]
94      )["rouge2"].mid
95
96      return {
97          "rouge2_precision": round(rouge_output.precision, 4),
98          "rouge2_recall": round(rouge_output.recall, 4),
99          "rouge2_fmeasure": round(rouge_output.fmeasure, 4),
100     }
101
102
103 training_args  = transformers.Seq2SeqTrainingArguments(
104     predict_with_generate=True,
105     evaluation_strategy="steps",
106     per_device_train_batch_size =config.batch_size,
107     per_device_eval_batch_size =config.batch_size,
108     output_dir=config.path_output,
109     warmup_steps=1000,
110     save_steps=5000,
111     logging_steps=1000,
112     eval_steps=5000,
113     save_total_limit =1,
114     learning_rate=5e-5,
115     adafactor=True,
116     fp16=True
117 )
118
119 trainer  = transformers.Seq2SeqTrainer(
120     model=tf2tf,
121     args=training_args,
122     compute_metrics=compute_metrics,
123     train_dataset=train_data,
124     eval_dataset=val_data,
125     tokenizer=tokenizer
126 )
127
128 trainer.train()
```

Quelltext 9.3: Trainingscode

Evaluationscode

```
# Imports
import datasets
import tf2tf_tud_gpu_config as config
import tf2tf_tud_gpu_helpers as helpers


# Main
tokenizer, tf2tf = helpers.load_tokenizer_and_model(from_checkpoint=True)

train_data, val_data, test_data = helpers.load_data(
    language=config.language,
    ratio_corpus_wiki=config.ratio_corpus_wiki,
    ratio_corpus_news=config.ratio_corpus_news,
    ratio_corpus_mlsum=config.ratio_corpus_mlsum,
    ratio_corpus_eng=config.ratio_corpus_eng
)

helpers.test_cuda()
helpers.explore_corpus(train_data)
helpers.empty_cache()
rouge = datasets.load_metric("rouge")

tf2tf = helpers.configure_model(tf2tf, tokenizer)
tf2tf.to("cuda")


def generate_summary(batch):
    inputs = tokenizer(
        batch["text"],
        padding="max_length",
        truncation=True,
        max_length=512,
        return_tensors="pt"
    )

    input_ids = inputs.input_ids.to("cuda")
    attention_mask = inputs.attention_mask.to("cuda")

    outputs = tf2tf.generate(input_ids, attention_mask=attention_mask)
    output_str = tokenizer.batch_decode(outputs, skip_special_tokens=True)
    batch["pred_summary"] = output_str

    return batch
```

```
44
45
46 results = test_data.map(
47     generate_summary,
48     batched=True,
49     batch_size=config.batch_size
50 )
51
52 print(
53     rouge.compute(
54         predictions=results["pred_summary"],
55         references=results["summary"],
56         rouge_types=["rouge2"]
57     )["rouge2"].mid
58 )
```

Quelltext 9.4: Evaluationscode

A Englischer Korpus

Text 1: Bericht über die Terroranschläge des 11. Septembers 2001.

The perpetrators knew what they were doing. The attacks were media-ready. The symbols they destroyed were precisely chosen: The White House and the Pentagon as symbols of power, the World Trade Center as a symbol of capital and business. 2,750 people, including 30 Germans, died in or on the towers of the WTC, 147 in the airplanes, 184 victims lost their lives in the Pentagon, 40 people were killed in the plane that crashed in Pennsylvania in the most perfidious terrorist attack yet unleashed by Islamist terrorists. The U.S. was in a state of shock: The attacks were the first quasi-military attack on the United States since 1814. America proved vulnerable at home on Sept. 11, 2001, for the first time since the struggles for its independence from England in the late 18th and early 19th centuries. As a result, the attacks had serious political and military consequences around the world. U.S. President George W. Bush, who was in Florida at the time of the attacks, made the following remarks in an initial televised address: “Make no mistake. We will hunt these people down and punish them to the end.“

Text 2: Bericht über das Finale der WM 2014.

German coach Joachim Löw saw no reason to change his starting eleven after the fantastic 7-1 triumph in the semifinals against host Brazil - until shortly before kickoff. Then the coach received bad news: Khedira had to pull out at short notice with calf problems, and Kramer was replaced in the starting lineup. The DFB team tried hard, but the South Americans were more determined. After a cross from Lavezzi, Higuain put in what was supposed to be a 1-0, but was correctly whistled back for an offside position. The longer the game lasted, the more both sides shied away from taking risks. The tempo of the match, which had been so racy in the first period, also dwindled with the courage. Instead, the duels became rougher: Mascherano and Aguero received warnings. And whenever a chance presented itself unexpectedly, a technical error crept in. It remained a tough struggle. Mascherano and Aguero, who had already been penalized, were lucky not to be sent off early with a yellow card when they first knocked Schweinsteiger down and then gave him a cut to the face. Schürrle put in a spirited spurt and found Götze in the center, who picked the ball off his chest and expertly fired in a left-footed volley - a dream goal at a very important time.

Text 3: Definition von Inflation.

Inflation is a sustained process of demonetization, which manifests itself in general price increases. One monetary unit can then constantly buy less, i.e. the purchasing power of money is permanently reduced. Inflation does not include one-time, temporary price level increases caused by unusual occurrences or price increases for specific goods or production factors. Inflation is measured by the increase in a price index that best reflects the general price level, such as the consumer price index for Germany. The percentage increase in the price index over a given period is referred to as the inflation rate. The quantity of money in the economy plays a particularly important role in the occurrence of inflation. If the quantity of goods in the economy as a whole is matched by an excessively large quantity of money, a condition for inflation is met. If the aggregate demand for goods exceeds the aggregate supply of goods, which cannot be increased in the short term, rising prices are the consequence and inflation sets in. The price increases trigger rising wages, and because of the higher income, the demand for goods increases. However, the higher wages also cause companies' costs to rise, which in turn leads to price increases for goods.

Text 4: Rechtsurteil zu Stornierungen während einer Pandemie.

If a traveler cancels the trip due to a virus pandemic, the tour operator is not entitled to compensation under Section 651 h (1) sentence 3 of the German Civil Code (BGB) if the traveler also cancels the trip. In addition, the lack of treatment options and vaccination justifies a right to cancel the trip free of charge. This decided the district court Stuttgart. The case was based on the following facts: A few days before the start of a multi-day flight and bus tour through Portugal in March 2020, the traveler declared the cancellation. She justified this with the spreading Corona virus. The tour operator also canceled the trip two days later. Nevertheless, it demanded payment of a cancellation fee from the traveler. Since the traveler refused to comply, a lawsuit was filed. The Stuttgart District Court ruled in favor of the traveler. The tour operator was not entitled to compensation pursuant to Section 651 h (1) sentence 3 BGB. On the one hand, the traveler could invoke extraordinary circumstances within the meaning of Section 651 h (3) of the German Civil Code. Secondly, it had to be taken into account that the tour operator had also cancelled the trip.

Text 5: Gebrauchsanweisung eines Kinderwagens.

To unfold the stroller, lay it flat on the ground, release the frame lock on the side and pull the handle up until the folding mechanism engages. Make sure the stroller is unfolded properly before use. To mount the rear axle, place it together with the brake system on the frame until the rear axle is audibly engaged on both sides. Hold the push button in the wheel hub and pull the wheel from the axle at the same time to remove the rear wheels and pull the wheel from the axle. To reassemble, press the push button and slide the wheel onto the axle until it locks into place. Before use, make sure that all wheels are properly mounted. The stroller is equipped with a parking brake on the rear axle. Press the brake pedal to lock the wheels. Lift the pedal to release the brake. Always block the brake when parking the stroller on uneven ground. The brake is not used to slow down when the stroller is moving and should only be locked when the stroller is at a complete standstill. The stroller is not suitable for walking.

Text 6: Erklärung von Algorithmen.

We encounter algorithms every day, both at work and in our leisure time, and it is impossible to imagine modern life without them. Mostly helpful but also not always harmless, algorithms are becoming increasingly important. What an algorithm is and how they shape our lives is explained in the following article. Generally speaking, an algorithm specifies a procedure for solving a problem. Based on this solution plan, input data is converted into output data in individual steps. Especially in computer science algorithms play an important role. They represent a basis of programming and are independent of a concrete programming language. Nevertheless algorithms are not to be found only in computer science or mathematics. Because algorithms are not only executed by a computer but can also be formulated and processed by humans in natural language. Our modern life is dependent on algorithms, without us always being aware of it. The areas of application of algorithms are very diverse: In the navigation system, they show us the shortest route, beat us as computer opponents in chess, control our sentence structure in Office Word or recommend a suitable partner for online dating. An algorithm can be found in many technical devices as well as behind our electronic communication.

B Deutscher Korpus

Text 1: Bericht über die Terroranschläge des 11. Septembers 2001.

Die Täter wussten, was sie taten. Die Anschläge waren mediengerecht umgesetzt. Die Symbole, die sie zerstörten, waren präzise ausgewählt: Das weiße Haus und das Pentagon als Symbole der Macht, das World Trade Center als Symbol des Kapitals und der Wirtschaft. 2.750 Menschen, darunter 30 Deutsche, starben in oder an den Türmen des WTC, 147 in den Flugzeugen, 184 Opfer kamen im Pentagon ums Leben, 40 Menschen wurden in dem in Pennsylvania abgestürtzten Flugzeug Opfer des bisher perfidesten Terroranschlags, den islamistische Terroristen ausgelöst haben. Die USA befanden sich im Schockzustand: Die Anschläge waren der erste quasi militärische Angriff auf die Vereinigten Staaten seit 1814. Amerika erwies sich am 11. September 2001 erstmals seit den Kämpfen um seine Unabhängigkeit von England im späten 18. und frühen 19. Jahrhundert als im eigenen Land verwundbar. Die Anschläge hatten daher weltweit gravierende politische und militärische Folgen. Der amerikanische Präsident George W. Bush, der sich zum Zeitpunkt der Anschläge in Florida aufhielt, äußert sich in einer ersten Fernsehansprache: “Täuschen Sie sich nicht. Wir werden diese Leute bis zum Ende jagen und bestrafen.“

Text 2: Bericht über das Finale der WM 2014.

Bundestrainer Joachim Löw sah nach dem famosen 7:1-Triumph im Halbfinale gegen Gastgeber Brasilien keinen Anlass, seine Startelf umzustellen - bis kurz vor Anpfiff. Dann erreichte den Bundestrainer die Hiobsbotschaft: Khedira meldete sich mit Wadenproblemen kurzfristig ab, für ihn rückte Kramer in die Startformation. Das DFB-Team agierte durchaus bemüht, doch zielstrebiger blieben die Südamerikaner. Nach einer Flanke von Lavezzi schob Higuain zum vermeintlichen 1:0 ein, wurde aber wegen einer Abseitsposition korrekterweise zurückgepfiffen. Je länger die Partie dauerte, desto mehr scheuten beide Seiten das Risiko. Mit dem Mut schwand auch das Tempo aus der im ersten Abschnitt noch so rassigen Begegnung. Dafür wurden die Zweikämpfe ruppiger: Mascherano und Aguero kassierten Verwarnungen. Und wenn sich mal unverhofft eine Chance bot, schlich sich ein technischer Fehler ein. Es blieb ein zähes Ringen. Die bereits vorbelasteten Mascherano und Aguero hatten Glück, dass sie nicht vorzeitig mit Gelb-Rot vom Platz mussten, als sie Schweinsteiger erst umsensten und dann eine Platzwunde im Gesicht verschafften. Zwei ließ das kalt: Schürrle zog noch einen beherzten Spurt an und fand Götze im Zentrum, der das Spielgerät mit der Brust runterpflückte und gekonnt mit links volley einschoss - ein Traumtor zu einem ganz wichtigen Zeitpunkt.

Text 3: Definition von Inflation.

Die Inflation ist ein anhaltender Prozess der Geldentwertung, der sich durch allgemeine Preiserhöhungen bemerkbar macht. Mit einer Geldeinheit kann dann ständig weniger gekauft werden, d.h. die Kaufkraft des Geldes vermindert sich dauernd. Nicht als Inflation gelten einmalige, vorübergehende, durch ungewöhnliche Vorkommnisse verursachte Preisniveauerhöhungen sowie Preissteigerungen für bestimmte Güter oder Produktionsfaktoren. Die Inflation wird gemessen am Anstieg eines das allgemeine Preisniveau am besten widerspiegelnden Preisindexes wie z.B. des Verbraucherpreisindexes für Deutschland. Der prozentuale Anstieg des Preisindexes in einem bestimmten Zeitraum wird als Inflationsrate bezeichnet. Beim Entstehen einer Inflation spielt besonders die Geldmenge in der Volkswirtschaft eine große Rolle. Steht der gesamtwirtschaftlichen Gütermenge eine zu große Geldmenge gegenüber, ist eine Bedingung für die Inflation gegeben. Übersteigt die gesamtwirtschaftliche Güternachfrage das gesamtwirtschaftliche Güterangebot, das kurzfristig nicht erhöht werden kann, sind steigende Preise die Folge und die Inflation setzt ein. Die Preissteigerungen lösen steigende Löhne aus, wegen des höheren Einkommens steigt die Nachfrage nach Gütern an. Die höheren Löhne bewirken jedoch auch steigende Kosten der Unternehmen, was wiederum zu Preissteigerungen für Güter führt.

Text 4: Rechtsurteil zu Stornierungen während einer Pandemie.

Storniert ein Reisender wegen einer Virus-Pandemie die Reise, so steht dem Reiseveranstalter kein Anspruch auf Entschädigung gemäß § 651 h Abs. 1 Satz 3 BGB zu, wenn er ebenfalls die Reise absagt. Zudem begründet die fehlende Therapiemöglichkeit und Impfung ein kostenloses Reiserücktrittsrecht. Dies hat das Amtsgericht Stuttgart entschieden. Dem Fall lag folgender Sachverhalt zugrunde: Wenige Tage vor Beginn einer mehrtägigen Flug- und Busrundreise durch Portugal im März 2020 erklärte die Reisende die Stornierung. Sie begründete dies mit dem sich ausbreitenden Corona-Virus. Die Reiseveranstalterin sagte zwei Tage später ebenfalls die Reise ab. Dennoch verlangte sie von der Reisenden die Zahlung einer Stornogebühr. Da sich die Reisende weigerte dem nachzukommen, kam es zu einem Klageverfahren. Das Amtsgericht Stuttgart entschied zu Gunsten der Reisenden. Der Reiseveranstalterin stehe kein Anspruch auf Entschädigung gemäß § 651 h Abs. 1 Satz 3 BGB zu. Denn zum einen könne sich die Reisende auf außergewöhnliche Umstände im Sinne von § 651 h Abs. 3 BGB berufen. Zum anderen sei zu berücksichtigen, dass die Reiseveranstalterin ebenfalls die Reise abgesagt hat.

Text 5: Gebrauchsanweisung eines Kinderwagens.

Lege den Kinderwagen zum Aufklappen flach auf den Boden, löse die Gestellsperre an der Seite und ziehe den Griff bis zum Einrasten des Klappmechanismus nach oben. Vergewissere Dich vor dem Gebrauch, ob der Kinderwagen richtig aufgeklappt ist. Um die Hinterachse zu montieren, setze diese samt Bremsanlage auf das Gestell, bis die Hinterachse beidseitig hörbar eingerastet ist. Halte den Druckknopf in der Radnabe gedrückt und ziehe das Rad gleichzeitig von der Achse, um die Hinterräder zu demontieren und ziehe das Rad von der Achse. Zur neuerlichen Montage drücke den Druckknopf und schiebe das Rad bis zum Einrasten auf die Achse. Versichere Dich vor dem Gebrauch, ob alle Räder richtig montiert sind. Der Kinderwagen ist mit einer Standbremse an der Hinterachse ausgestattet. Betätige das Bremspedal, um die Räder zu blockieren. Hebe das Pedal an, um die Bremse zu lösen. Blockiere die Bremse immer, wenn Du den Kinderwagen auf unebenem Untergrund abstellst. Die Bremse dient nicht zur Verlangsamung bei der Fahrt und ist ausschließlich bei völligem Stillstand des Kinderwagens zu blockieren. Der Kinderwagen ist nicht zum Laufen geeignet.

Text 6: Erklärung von Algorithmen.

Algorithmen begegnen uns täglich sowohl auf der Arbeit als auch in der Freizeit und sind aus unserem modernen Leben nicht mehr wegzudenken. Meist hilfreich aber auch nicht immer unbedenklich, kommen Algorithmen immer größere Bedeutung zu. Was ein Algorithmus ist und wie sie unser Leben prägen, wird in dem folgenden Artikel erläutert. Allgemein gesagt, gibt ein Algorithmus eine Vorgehensweise vor, um ein Problem zu lösen. Anhand dieses Lösungsplans werden in Einzelschritten Eingabedaten in Ausgabedaten umgewandelt. Besonders in der Informatik spielen Algorithmen eine große Rolle. Sie stellen eine Grundlage der Programmierung dar und sind unabhängig von einer konkreten Programmiersprache. Trotzdem sind Algorithmen nicht nur in der Informatik oder Mathematik vorzufinden. Denn Algorithmen werden nicht nur maschinell durch einen Rechner ausgeführt sondern können auch von Menschen in natürlicher Sprache formuliert und abgearbeitet werden. Unser modernes Leben ist abhängig von Algorithmen, ohne dass uns dies immer bewusst ist. Die Einsatzgebiete von Algorithmen sind sehr vielfältig: Im Navi zeigen sie uns den kürzesten Weg, schlagen uns als Computergegner im Schach, kontrollieren unseren Satzbau in Office Word oder empfehlen uns einen passenden Partner beim Online-Dating. Ein Algorithmus steckt in vielen technischen Geräten sowie hinter unserer elektronischen Kommunikation.