Hochschule für Technik und Wirtschaft Dresden
Fakultät Informatik/ Mathematik

Abschlussarbeit zur Erlangung des akademischen Grades

Master of Science, M. Sc.

Thema:

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

eingereicht von:	Daniel Vogel
eingereicht am:	8. August 2021
Erstgutachter:	Prof. Dr. Hans-Joachim Böhme
Zweitgutachter:	Dipl.-Kfm. Torsten Rex

Abstract

Vogel, Daniel: Adaption multilingual vortrainierter Modelle zur automatischen Zusammenfassung von Texten auf die deutsche Sprache, Hochschule für Technik und Wirtschaft Dresden, Fakultät Informatik/ Mathematik, Studiengang Angewandte Informatik, Studienrichtung Data Science, Masterarbeit, 2021.

62 Seiten, 51 Literaturquellen, 2 Anhänge.

In der vorliegenden Arbeit wird die Adaption multilingual vortrainierter Modelle zur automatischen Zusammenfassung von Texten auf die deutsche Sprache erforscht und demonstriert. Hierfür werden entsprechende Grundlagen in Deep Learning und Natural Language Processing dargestellt. Dabei ist insbesondere der kontextbezogene Fortschritt durch Transfer Learning in Verbindung mit Deep Language Representations hervorzuheben. Es schließen sich verschiedene Experimente an, deren Architektur und Datengrundlage zuvor methodisch aufbauend definiert wird.

Die automatische Zusammenfassung von Texten lässt sich mithilfe eines Sequence-to-Sequence-Transformer-Modells, welches über einen vortrainierten Encoder und Decoder verfügt, SOTA-konform realisieren, insbesondere unter Nutzung von BERT und BART. Die Adaption auf die deutsche Sprache bedarf nicht zwingend einer architektonischen Anpassung, sondern einem Austausch der Textdaten in der entsprechenden Zielsprache, um Zusammenfassungen auf SOTA-Niveau zu generieren. Die Qualität der sprachtechnischen Adaption ist sehr stark von der Qualität der vortrainierten Modelle sowie dem Umfang und der Beschaffenheit der zugrundeliegenden Textdaten abhängig.

Thesen

1. Die automatische Zusammenfassung von Texten lässt sich mithilfe eines Sequence-to-Sequence-Transformer-Modells, welches über einen vortrainierten Encoder und Decoder verfügt, SOTA-konform realisieren, insbesondere unter Nutzung von BERT und BART.

2. Die Adaption auf die deutsche Sprache bedarf nicht zwingend einer architektonischen Anpassung, sondern einem Austausch der Textdaten in der entsprechenden Zielsprache, um Zusammenfassungen auf SOTA-Niveau zu generieren.

3. Die Qualität der sprachtechnischen Adaption ist sehr stark von der Qualität der vortrainierten Modelle sowie dem Umfang und der Beschaffenheit der zugrundeliegenden Textdaten abhängig.

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
DLR	Deep Language Representations
ELMo	Embeddings from Language Models
GBERT	German BERT
GloVe	Global Vectors for Word Representation
GPT	Generative Pre-Trained Transformer
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

SDPA Scaled-Dot-Product-Attention

SOTA State-of-the-Art

SYSZ Systemzusammenfassung

TF2TF Sequence-to-Sequence-Transformer-Modell

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. Einsatzmöglichkeiten entsprechender ATS-Modelle sind beispielsweise die Zusammenfassung von Nachrichten und Protokollen 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.

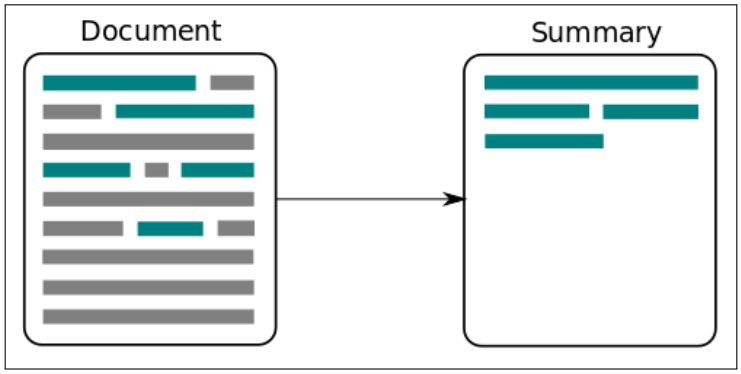

Abbildung 1.1: Ablauf einer automatischen Zusammenfassung [Thaker, 2019].

1.1 Motivation

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. Außerdem können Pflegeroboter, welche mitunter demente Patienten betreuen, durch ein ATS-Modell mit notwendigem Kontextwissen für die anstehenden Gespräche ausgestattet werden. Somit werden die Gespräche im Sinne der Behandlung persönlicher und erfolgreicher.

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 im Rahmen von Machine Learning (ML) 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 als Paraphrasierung bezeichnet. Auch Menschen müssen diese Fähigkeit erst einmal schwerlich erlernen.

1.2 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 dargestellten 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 existierende Modelle auf eine andere Sprache adaptiert werden?
- Wie qualitativ und skalierbar ist die Lösung?

1.3 Aufbau der Arbeit

Nach der Einleitung werden zunächst die Grundlagen des Deep Learning (DL) und des NLP dargestellt. Im Kapitel des DL werden neuronale Netze als solches definiert und relevante Architekturen 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 Deep Language Representations thematisiert. Bevor die bis dahin behandelten Komponenten in ein tatsächliches Modell integriert und adaptiert werden können, ist die weitergehende Beschreibung des Forschungsstandes und der Datengrundlage erforderlich. Nachfolgend werden verschiedene Experimente durchgeführt und evaluiert, welche unter anderem die Reproduktion des State-of-the-Art (SOTA) und die Adaption auf die deutsche Sprache vorsehen. Der entsprechende Quellcode wird in Python entwickelt. Abbildung 1.2 stellt den Aufbau der Arbeit dar. Hier werden gleichzeitig die kapitelübergreifenden Zusammenhänge deutlich.

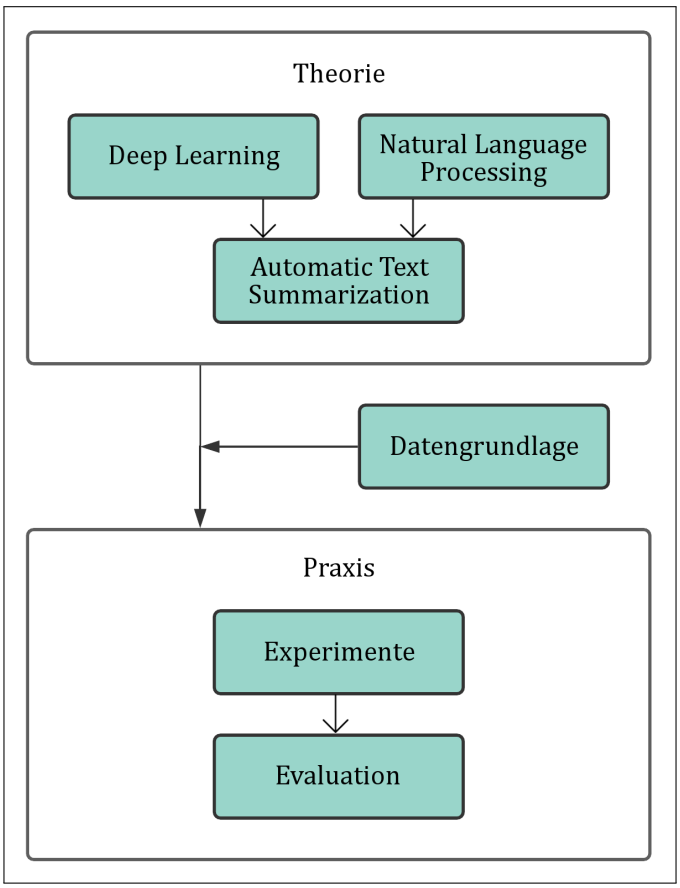

Abbildung 1.2: Aufbau der Arbeit.

1.4 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 waren 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 SOTA, denn um den SOTA konkurrieren fast ausschließlich sogenannte Transformer [Yang et al., 2019, Rothe et al., 2020]. Diese basieren auf den Encoder-Decoder-Architekturen, implementieren verschiedenartige Attention-Mechanismen und konnten sich qualitativ beweisen. Diese Arbeit wird daher ebenfalls diesen Ansatz verfolgen, darüber hinaus jedoch die bislang unzulänglich behandelte Adaption auf die deutsche Sprache behandeln.

Weiterhin hat der Durchbruch frei verfügbarer vortrainierter Modelle die NLP-Welt revolutioniert, wie beispielsweise Bidirectional Encoder Representations from Transformers (BERT) sowie diverse Weiterentwicklungen [Devlin et al., 2019]. Verschiedenste NLP-Aufgaben wie die ATS konnten hiervon sehr stark profitieren. Die konkreten Funktionsweisen werden im Verlauf dieser Arbeit vertieft, wenn entsprechende Grundlagen kenntlich gemacht wurden. Dort werden der Forschungsstand der ATS und vergleichbare Arbeiten separat thematisiert sowie entsprechende Verbesserungen hervorgehoben.

2 Deep Learning

Deep Learning ist ein Teilbereich des ML. ML-Algorithmen analysieren Daten automatisiert mittels mathematischer Methoden der Mustererkennung [Khanna, 2019, S. 455-457]. DL-Algorithmen bedienen sich hingegen vielschichtiger und hoch parametrisierter neuronaler Netze, um dem menschlichen Gehirn bestmöglich nachzuempfinden und Anwendungen mit künstlicher Intelligenz auszustatten [Goodfellow et al., 2016, S. 1-2]. 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 beabsichtigte Tiefe. Hier werden Informationen weiterverarbeitet, abstrahiert und reduziert [Goodfellow et al., 2016, S. 164-165]. Der Aufbau neuronaler Netze sowie deren Funktionsweise und ausgewählte Architekturen werden in diesem 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 ausreichend Signale registriert wurden. Diese Aktivierung sendet folglich weitere Signale an andere Neuronen, wie Abbildung 2.1 im technischen Sinne exemplarisch skizziert. Hierfür werden Aktivierungsfunktionen benötigt, welche die gewichteten Eingangssignale in ein Ausgangssignal konvertieren. Sie ermöglichen es, nichtlineare Zusammenhänge zwischen den Eingangs- und den Ausgangsdaten herzustellen [Zhang et al., 2020, S. 134].

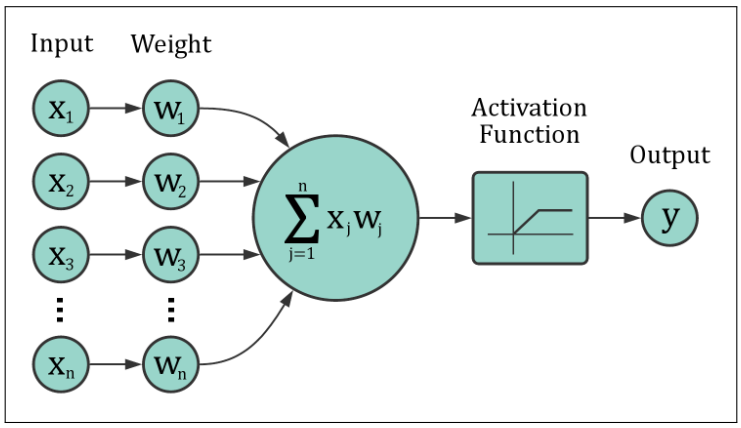

Abbildung 2.1: Aufbau eines künstlichen Neurons [McCullum, 2020].

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 sind.

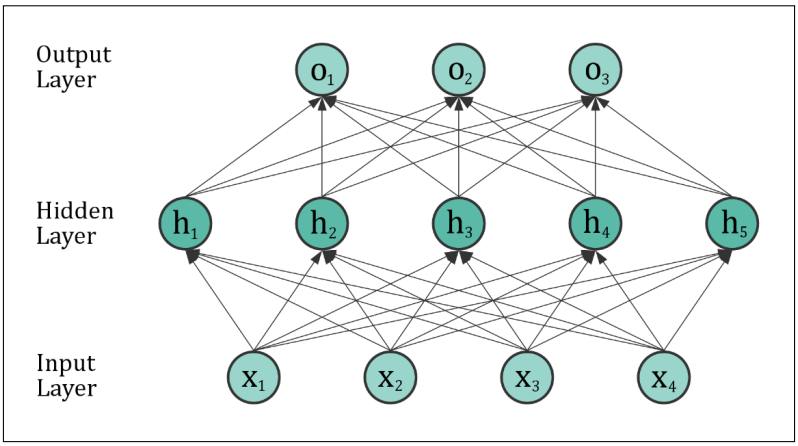

Abbildung 2.2: Aufbau eines MLP [Raschka et al., 2019, S. 388].

Ziel der hoch parametrisierten neuronalen Netze ist es, komplexe Polynomfunktionen höheren Grades bestmöglich zu approximieren und so verschiedenste Probleme zu lösen. Der Grad einer Polynomfunktion wird an der höchsten Potenz des jeweiligen Funktionsterms gemessen.

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. 167-169].

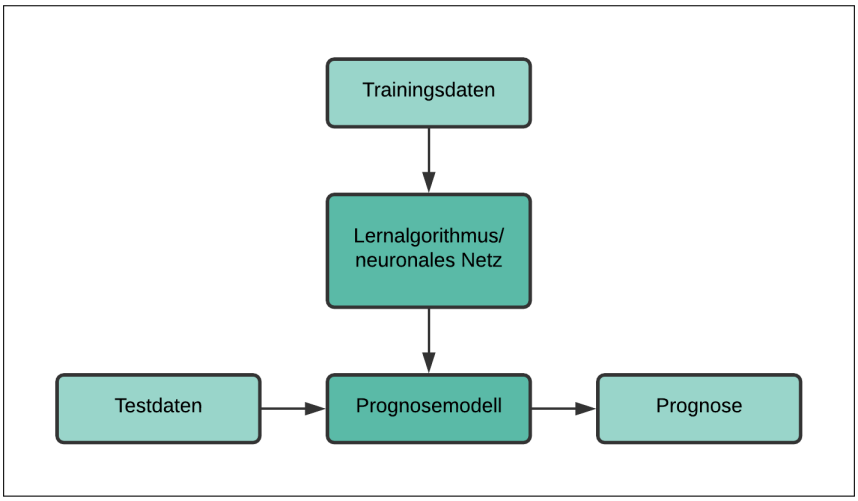

Abbildung 2.3: Supervised Learning [Raschka et al., 2019, S. 3].

Der Trainingsprozess erfolgt über mehrere sogenannte Epochen. Hier werden dem neuronalen Netz die Eingangsdaten zugeführt und beidseitige Propagationen ausgeführt. Hierbei ist wichtig, kein Over- oder 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 [Goodfellow et al., 2016, S. 108-110]. Es gibt verschiedene Ansätze, um beispielsweise Overfitting vorzubeugen. Hier seien Batch Normalization, Dropout und Early Stopping genannt, wobei entsprechende Mechanismen an anderweitiger Stelle erläutert werden [Goodfellow et al., 2016, S. 241, 255, 276, 313].

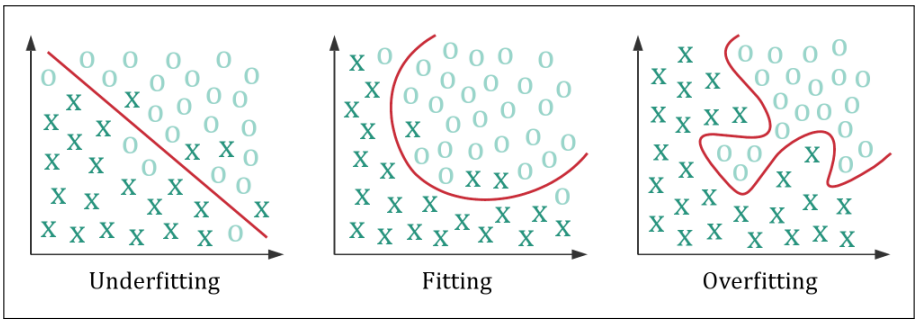

Abbildung 2.4: Typen von Generalisierungseffekten [Edpresso, O. J.].

2.2 Architekturen

Um die ATS mithilfe neuronaler Netze zu modellieren, werden nun ausgewählte Architekturen vorgestellt. Diese gehen weit über die als Grundlage beschriebenen MLP hinaus. Eingangsdaten der ATS haben in jedem Fall einen Textcharakter. Dabei ist die Reihenfolge der Sätze und der Wörter von großer Bedeutung, um Texte hinreichend verstehen und anschließend generieren zu können. Dies wird von den nachfolgend beschriebenen Architekturen gewährleistet [Zhang et al., 2020, S. 301].

2.2.1 Encoder-Decoder-Networks

Encoder-Decoder-Architekturen sind zunächst einmal als Template zu verstehen, welches einer stets individuellen Entwicklung bedarf. Dabei bestehen entsprechende Modelle aus einem Encoder und einem Decoder. Beide Module bestehen aus neuronalen Netzen, welche beispielsweise durch RNN oder auch LSTM repräsentiert werden können. Hierdurch würde zugleich die Verarbeitung sequenzieller Daten ermöglicht. Hinsichtlich der anvisierten ATS spricht man daher auch von Sequence-to-Sequence-Modellen [Vaswani et al., 2017, S. 2].

Im Encoder wird die Eingabesequenz zuerst eingebettet. Dabei 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.

Der Decoder wird mit eben diesen Ausgangsdaten des Encoders initialisiert. Die entsprechende Zustandsrepräsentation wird ebenfalls mithilfe eines neuronalen Netzes verarbeitet [Vaswani et al., 2017, S. 2]. 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

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

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 [Vaswani et al., 2017, S. 2].

Um die theoretisch und abstrakt beschriebene Architektur zu veranschaulichen, werden die wesentlichen Module nun abschließend in Abbildung 2.5 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, S. 2].

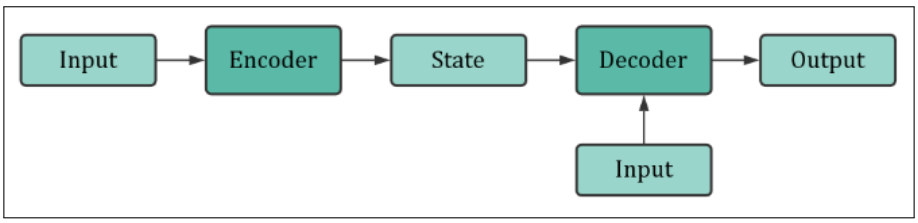

Abbildung 2.5: Encoder-Decoder-Architektur [Zhang et al., 2020, S. 375].

2.2.2 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. Dabei geht es der Übersetzung folgend 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 der nachfolgend beschriebenen Scaled-Dot-Product-Attention (SDPA) 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 [Vaswani et al., 2017, S. 1]. Das ATS-Modell soll mathematisch also menschenähnlichem Verhalten nachempfinden [Zhang et al., 2020, S. 389].

Die SDPA besteht hierfür aus einer Attention-Funktion, welche Queries und eine Menge von Key-Value-Paaren verarbeiten kann. Hierbei gehen Queries und Keys der Dimension d_k sowie Values der Dimension d_v ein. Die SDPA kann somit gemäß

$$Attention(Q, K, V) = Softmax(\frac{Q \cdot K^T}{\sqrt{d_k}}) \cdot V$$

berechnet werden, indem das Skalarprodukt der Queries und Keys berechnet, durch einen dimensionsabhängigen Term dividiert und unter Anwendung einer Softmax-Funktion mit den Values multipliziert wird [Vaswani et al., 2017, S. 4].

Die SDPA kann innerhalb einer Encoder-Decoder-Architektur wie folgt an einem Index t integriert werden. Der Encoder bettet 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 diese Query wiederum durch den vorangegangenen verborgenen Zustand des Decoders repräsentiert wird. 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, S. 5].

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 2.6 visualisiert diese Self-Attention.

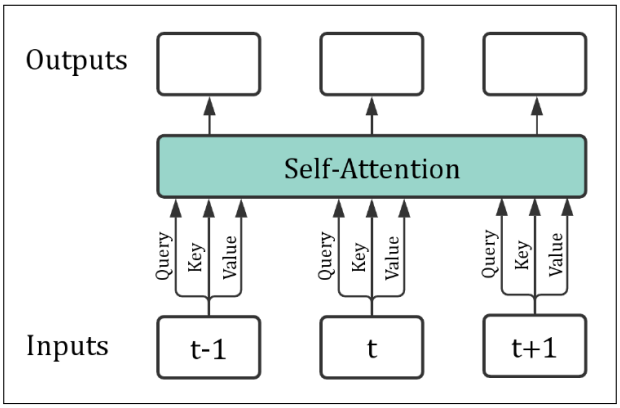

Abbildung 2.6: Self-Attention [Zhang et al., 2020, S. 400].

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 2.7 visualisiert die Multi-Head-Attention architektonisch.

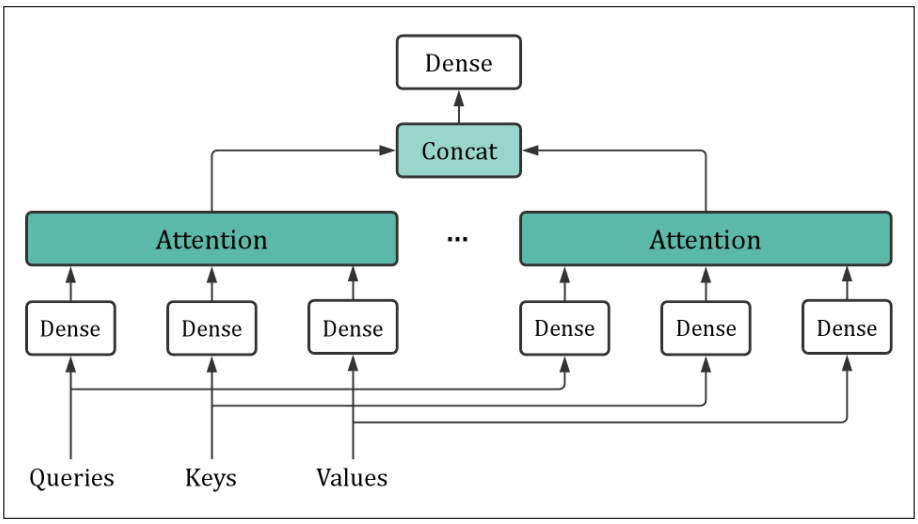

Abbildung 2.7: Multi-Head-Attention [Zhang et al., 2020, S. 400].

2.2.3 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 [Vaswani et al., 2017, S. 5-6].

Architektonisch werden die 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 2.8 visualisiert die beschriebene Architektur.

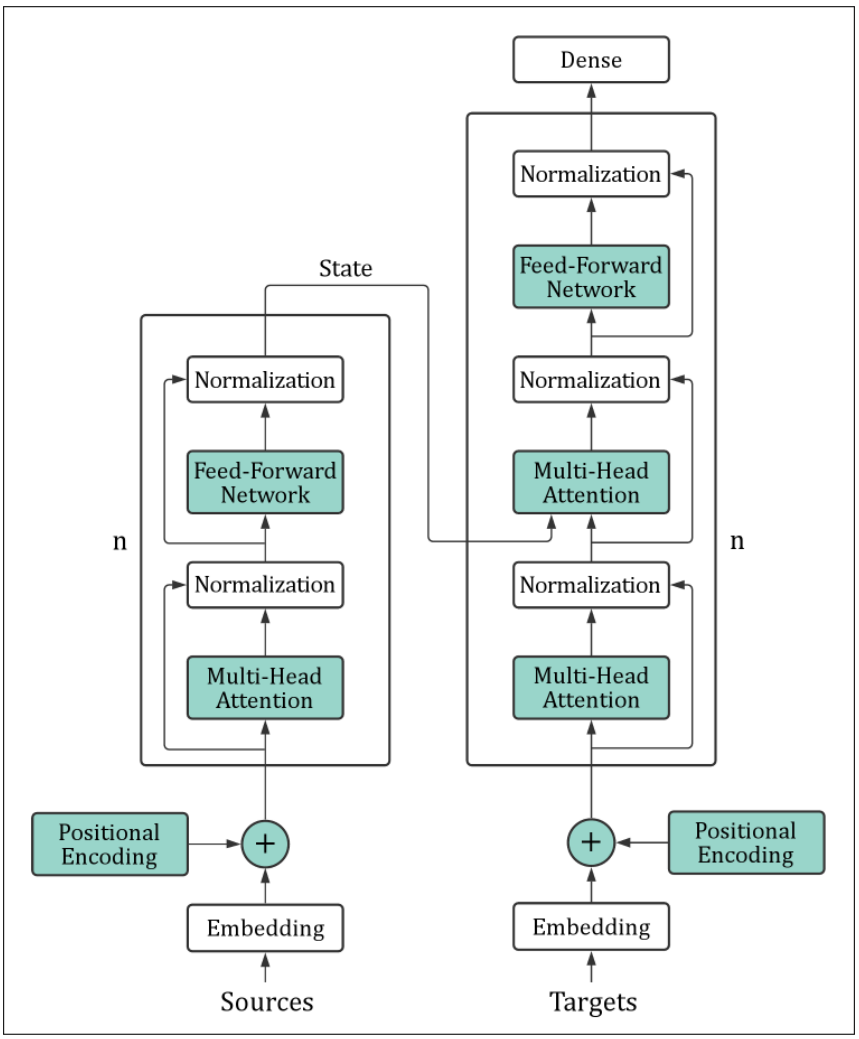

Abbildung 2.8: Transformer-Architektur [Vaswani et al., 2017, S. 3].

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, 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 [Goodfellow et al., 2016, S. 208]. 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 2.9 verdeutlicht diese Bedingungen. Ziel ist allgemein eine möglichst schnelle Konvergenz.

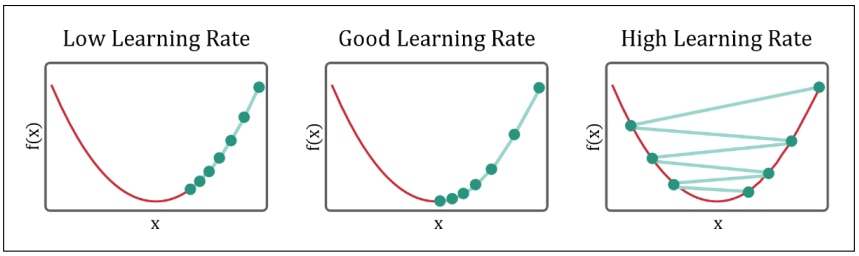

Abbildung 2.9: Konvergenzverhalten im Gradientenverfahren [Zhang et al., 2020, S. 429].

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 [Goodfellow et al., 2016, S. 290]. Die Größe der Stichprobe wird üblicherweise als Batch Size bezeichnet und an dieser Stelle nur als weitergehender Hyperparameter genannt.

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 2.10 zeigt. Das Momentum ist somit potenziell in der Lage, lokale Minima zu überspringen und die Suche erst im tatsächlichen globalen Minimum zu beenden [Goodfellow et al., 2016, S. 292-295].

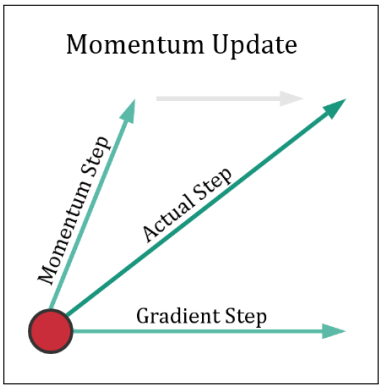

Abbildung 2.10: Gradientenverfahren unter Einfluss eines Momentums [CS231N, O. J.].

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. 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 2.11 veranschaulicht. Zudem ist ein gezieltes weitergehendes Training (engl. Fine-Tuning) mit entsprechenden Daten notwendig. Inwieweit die neuen Daten auf die vortrainierten Modelle einwirken sollen, ist individuell zu erproben [Goodfellow et al., 2016, S. 319, 534].

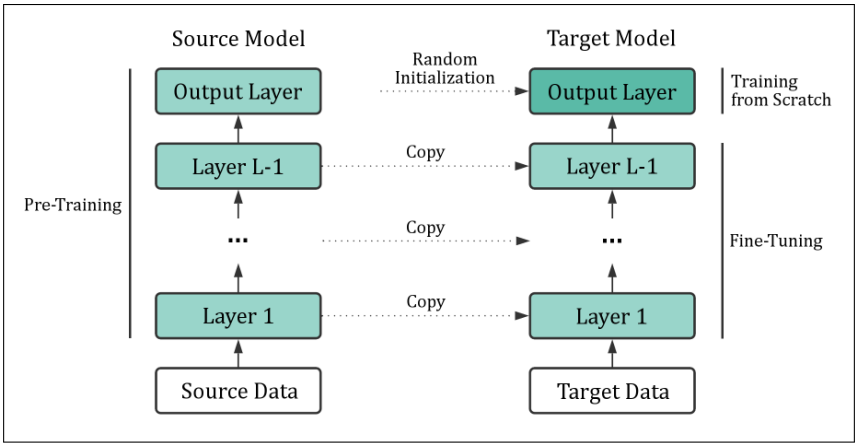

Abbildung 2.11: Fine-Tuning vortrainierter Modelle [Zhang et al., 2020, S. 555].

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 Zuführung massenhafter Daten
- Reduktion von Anforderungen, Kosten und Stromverbrauch beim Fine-Tuning

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 [Goodfellow et al., 2016, S. 456]. 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.

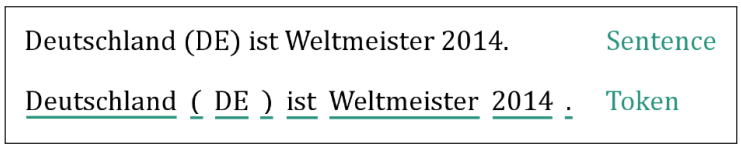

Abbildung 3.1: Tokenisierung eines beispielhaften Satzes.

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.

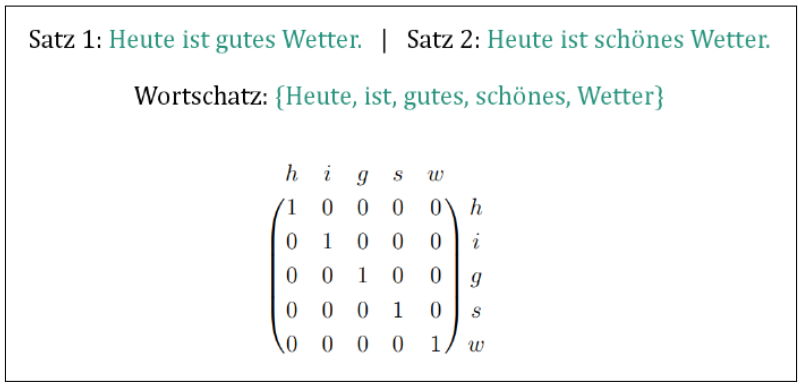

Abbildung 3.2: One-Hot-Encoding mit zwei beispielhaften Sätzen.

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.

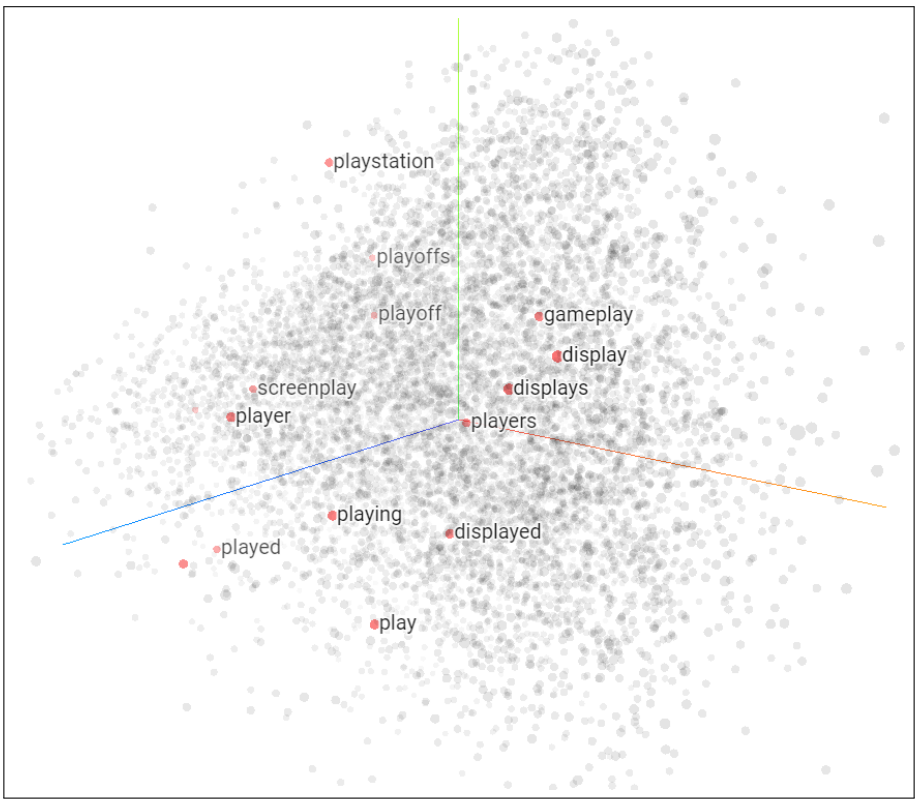

Abbildung 3.3: Word2Vec mit dem Embedding Projector von TensorFlow.

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, während die komprimierende Funktion kleinere Modellgrößen verursacht [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 (DLR), 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 DLR 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].

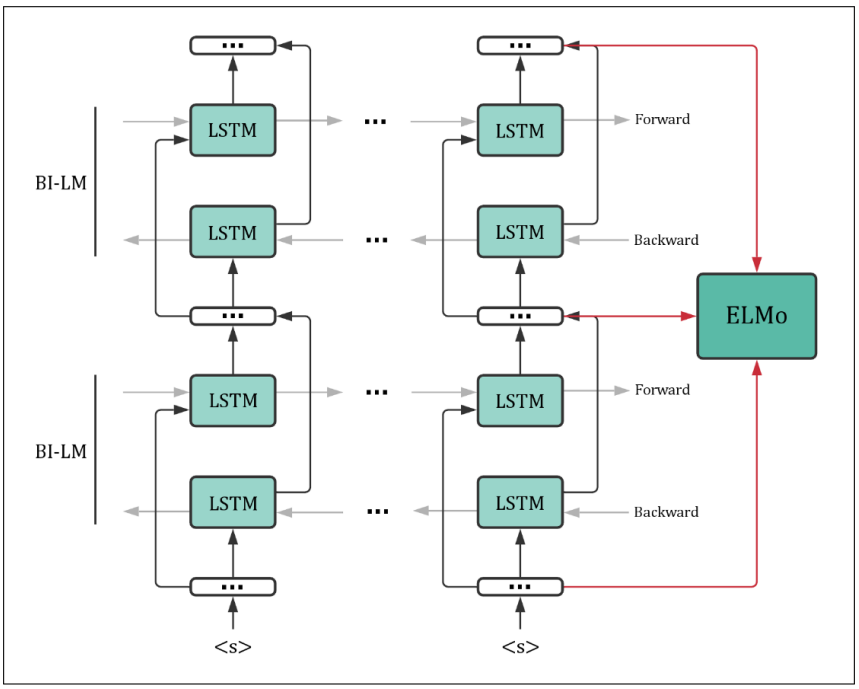

Abbildung 3.4: Architektur und Funktionsweise von ELMo [Irene, 2018].

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 40 GB 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 DLR 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 zumeist 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].

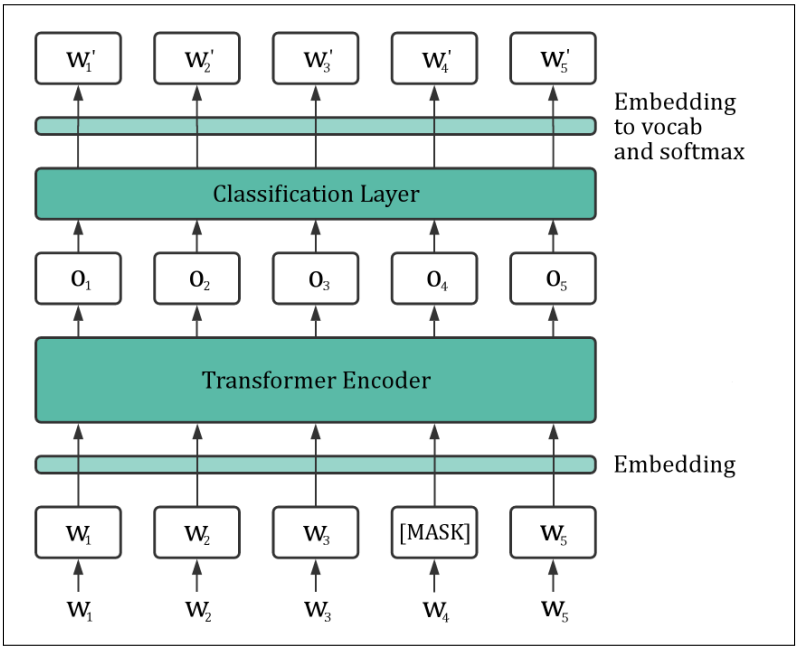

Abbildung 3.5: Architektur von BERT mit MLM [Devlin et al., 2019, S. 3].

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].

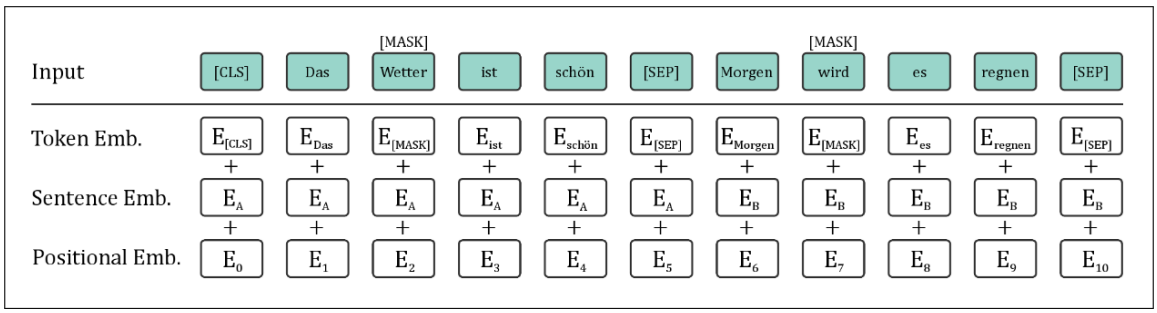

Abbildung 3.6: Repräsentation von Textdaten mithilfe von BERT [Devlin et al., 2019, S. 3].

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-R

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 aus dem Jahre 2019, welche sich insbesondere für das Vortraining von Sequence-to-Sequence-Modellen eignet. Hierfür wird eingehender Text mithilfe einer Rauschfunktion verändert, ähnlich wie beim bidirektionalen Maskieren in BERT. Das Modell versucht daraufhin, den ursprünglichen Text zu rekonstruieren, um unüberwacht einen Lernfortschritt zu erzielen. Somit ist BART nicht auf bestimmte NLP-Aufgaben limitiert, sondern vielseitig einsetzbar, wenngleich die Entwickler die besondere Eignung für NLG-Aufgaben unter der Bedingung eines entsprechenden Fine-Tunings hervorheben [Lewis et al., 2019, S. 1].

Wiederholend und vergleichend sei aufgeführt (siehe Abbildung 3.7): BERT maskiert eingehenden Text bidirektional, bevor fehlende Token unabhängig davon prognostiziert werden. Dabei eignet sich BERT insbesondere als Encoder, während er die NLG-Fähigkeiten eines Decoders erst schwerlich erlernen muss. GPT prognostiziert fehlende Token indes autoregressiv, jedoch nicht bidirektional. Daher eignet sich GPT insbesondere als Decoder. BART bedient sich nun den wesentlichen Vorteilen der beiden genannten DLR. Demnach wird eingehender Text bei BART ebenfalls bidirektional maskiert und prognostiziert Encoder). Anschließend wird die Wahrscheinlichkeit des eingegangenen Textes autoregressiv berechnet (Decoder). In einem weiterführenden Fine-Tuning sind hingegen nur unberauschte Texte zu verwenden, da die bereits erlernten verborgenen Zustände der DLR zum Einsatz kommen. BART generalisiert also bisherige Architekturen wie BERT oder auch GPT, weshalb es sich weiterhin um eine Transformer-Architektur handelt [Lewis et al., 2019, S. 2-3].

Die besten Ergebnisse erzielte BART in der jüngsten Vergangenheit bei eben genau NLG-Aufgaben wie der ATS, sowohl mono- als auch multilingual. Dabei schließt BART qualitativ nicht nur zu BERT und XLM-R auf, sondern übersteigt sie zu Teilen sogar [Lewis et al., 2019].

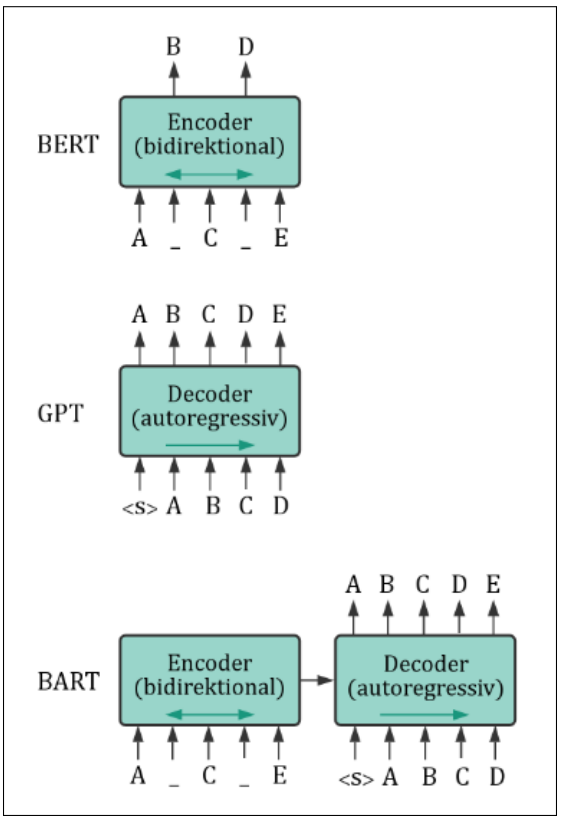

Abbildung 3.7: BERT, GPT und BART im Vergleich [Lewis et al., 2019, S. 2].

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 im Kontext dieser Arbeit wird der BLEU-Score 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 eine Architektur dargestellt.

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, die Architektur 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 vorgestellt. Diese ist als Sequence-to-Sequence-Transformer-Modell (TF2TF) zu verstehen. Dabei wird sowohl der Encoder als auch der Decoder durch eine eigenständige gemäß TL vortrainierte DLR 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 einer autarken DLR 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 einer DLR zu initialisieren, welche 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].

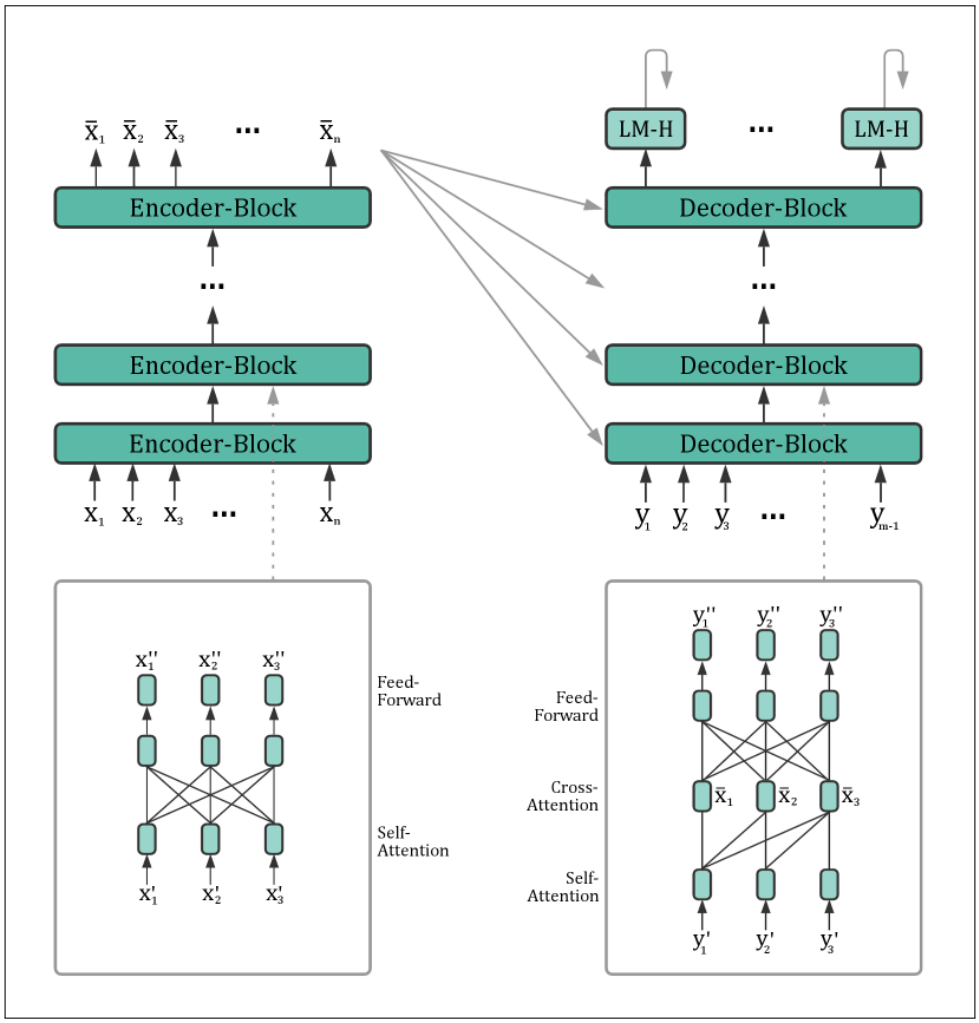

Abbildung 4.1: Sequence-to-Sequence-Transformer-Modell mit BERT [Von Platen, 2020].

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 ursprünglichen 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.

4.4 Adaption der Sprache

Die oben beschriebene 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 sprachübergreifend verborgene Strukturen erlernt, um anschließend monolingual davon zu profitieren. Zudem wird dem Problem entgegengewirkt, dass sprachintern zu wenig Textdaten zur Verfügung stehen [Moberg, 2020].

DLR, welche organisationsextern bereits multilingual vortrainiert und bereitgestellt wurden, sind unter anderem die multilingualen Versionen von BERT und XLM-R. Hinsichtlich einer anschließenden deutschsprachigen Nutzung in der ATS ist folglich zwingend ein entsprechendes sprachbezogenes Training erforderlich [Yang et al., 2019, S. 3-4]. Die Sprache der in diesem Fine-Tuning zugeführten Textdaten bestimmt stets die Zielsprache des entstehenden ATS-Modells. Die Adaption gilt also nicht nur für die deutsche Sprache.

Dennoch gibt es verschiedene Eigenarten in der deutschen Sprache, welche zu berücksichtigen sind. Die Syntax unterliegt demnach unzähligen Ausnahmen, welche mitunter eine Uneindeutigkeit der Sprache bewirken. Zudem ist es möglich, lange und somit teilweise noch unbekannte Komposita zu konstruieren. Darüber hinaus werden Elemente anderer Sprachen immer weiter in die deutsche Sprache integriert, wie etwa Anglizismen. Dabei werden die Grenzen der einzelnen Sprachen immer unkenntlicher, was modellseitig zu Ungenauigkeiten führen kann. Nicht zuletzt verfügt die deutsche Sprache über Umlaute, welche eine entsprechende Unterstützung erfordern [Sprachenfabrik, 2016].

Ähnlich wie die sprachtechnische Adaption kann auch eine domänenspezifische Adaption erfolgen. Dies geht aber womöglich über den Austausch der Textdaten hinaus und bedarf mitunter architektonischen Anpassungen, welche das NLU und die NLG in der Zieldomäne begünstigen.

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 englisch- 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.

5.1 Datenauswahl

Unter Berücksichtigung obiger Anforderungen werden nun vier Korpora ausgewählt. Diese werden wie folgt deklariert und nachfolgend beschrieben: K_{eng}, K_{wik}, K_{nws}, K_{mls}. Der Korpus K_{eng} 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 nachrichtenorientierten 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 [Rothe et al., 2020, S. 6].

Die anderen drei Korpora dienen dem Fine-Tuning und bestehen demzufolge aus deutschsprachigen Textdaten. Diese wurden teilweise selbst zusammengestellt oder vorverarbeitet. Der Korpus K_{wik} wurde 2019 im Kontext der Swiss Text Analytics Conference als Grundlage eines Wettbewerbs 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_{nws} wurde durch einen in Python selbst entwickelten Crawler generiert. In einer Zeitspanne von über zehn Monaten wurden mehr als 50.000 Nachrichtenartikel automatisiert kollektiert [Bird et al., 2009, S. 79, 83, 416]. Diese entstammen den folgenden Nachrichtenagenturen: SÜDDEUTSCHE ZEITUNG, DER SPIEGEL, ZEIT ONLINE. Nach Sichtung der verfügbaren Daten können nur die Artikel der ZEIT ONLINE als geeignet bewertet werden. Die Texte der anderen beiden Agenturen lassen sich technisch nicht erfassen. Folglich sind etwa 15.000 Datensätze nutzbar. Der Korpus K_{mls} nennt sich MLSUM und steht als multilingualer Korpus für das Training der ATS zur Verfügung. Die darin enthaltenen über 200.000 deutschsprachigen Datensätze werden a priori extrahiert und entstammen erneut einer vornehmlich nachrichtenorientierten Domäne [Scialom et al., 2020].

Um insbesondere der Kritik an den präsentierten Metriken entgegenzuwirken und belastbare Aussagen treffen zu können, werden nun ein englisch- und ein deutschsprachiger Korpus eingeführt. Diese dienen der qualitativen Analyse maschinell generierter Zusammenfassungen (siehe Anhang A und B). Dabei ist strukturell und inhaltlich von korpusübergreifender Gleichheit auszugehen, während die Texte korpusintern möglichst

unterschiedlich sind. Neben Berichten und einer Definition ist beispielsweise auch ein Rechtsurteil und eine Anleitung enthalten. Die Texte können hierbei der Allgemeinsprache, der Fachsprache und der Alltagssprache zugeordnet werden. Die Texte entstammen bundesweiten Informations- sowie Nachrichtenkanälen und wurden entsprechend übersetzt. Ergänzend ist jedem Text eine manuell verfasste Zusammenfassung zugeordnet, welche als menschlicher Gold-Standard bezeichnet wird [Wissler et al., 2014].

5.2 Datenexploration

Um hinreichende Kenntnis über die vorliegenden Textdaten zu erlangen und bedarfsorientierte Vorverarbeitungsschritte ableiten zu können, ist eine entsprechende Exploration notwendig. Der Quellcode ist dem Anhang zu entnehmen. Demnach bestehen englischsprachige Texte durchschnittlich aus etwa 850 Wörtern, Zusammenfassungen hingegen aus etwa 60 Wörtern. Dies spricht für einen hohen Abstraktionsgrad und eine hohe Verdichtung. Texte der deutschsprachigen Korpora bestehen durchschnittlich aus etwa 500 Wörtern, Zusammenfassungen hingegen aus etwa 30 Wörtern. Die häufigsten Uni-, Bi- und Trigramme verhalten sich erwartungsgemäß und sind Tabelle 5.1 zu entnehmen. Diese Informationen genügen vorerst in Verbindung mit einem sorgfältigen Blick in eine Stichprobe der vorliegenden Texte.

Unigramme	**Bigramme**	**Trigramme**
der	in der	in den vergangenen
die	in den	in den USA
und	mit dem	er in der
in	in die	an der Universität
den	an der	sich in der
das	für die	in der ersten

Tabelle 5.1: Übersicht der häufigsten N-Gramme der deutschen Korpora.

5.3 Hinweise

Üblicherweise existiert beim anvisierten Training die Gefahr der sogenannten Exploitation. Diese Gefahr meint im Kontext der ATS konkret, dass das zugrundeliegende Modell die Struktur anstatt der Inhalte der Artikel lernt [Goodfellow et al., 2016, S. 476]. 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 bereits während des Ladeprozesses ignoriert. Dennoch sollte zur Vorbeugung stets eine Mischung aus den drei deutschsprachigen Korpora vorgenommen werden [Bird et al., 2009, S. 42].

6 Experimente

Unter Kenntnis der Grundlagen des DL, des NLP und der ATS wird nun ein Ablauf von Experimenten konzipiert, welcher verschiedene TF2TF definiert und ein entsprechendes Fine-Tuning vorsieht. 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 A100) vollends auszuschöpfen [ZIH, 2021]. Der Quellcode ist dem Anhang zu entnehmen.

6.2 Reproduktion auf englischen Daten

Zunächst erfolgt die Reproduktion des SOTA 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 TF2TF unter Nutzung von BERT und XLM-R als Encoder und Decoder auf den englischsprachigen Daten (K_{eng}) 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. Dies erfolgt bekanntermaßen über den Austausch der Textdaten in die Zielsprache, hier Deutsch. Dafür werden zwei weitere TF2TF unter Nutzung der oben genannten DLR jeweils auf Basis aller deutschsprachigen Daten ($K_{wik} \cup K_{nws} \cup K_{mls}$) trainiert und evaluiert.

6.4 Adaption auf multilingualen Daten

Basierend auf der Annahme, multilinguale DLR würden von verborgenen Strukturen anderer Sprachen profitieren, werden erneut zwei TF2TF unter Nutzung beider DLR jeweils auf einem Mix aus allen englisch- und deutschsprachigen Daten ($K_{eng} \cup K_{wik} \cup K_{nws} \cup K_{mls}$) trainiert und auf Basis einer deutschsprachigen Evaluation mit den bisherigen Fortschritten verglichen. BART wird nun ebenfalls herangezogen, allerdings nicht selbst trainiert, sondern fortan nur als multilingual weitertrainiertes Modell von HuggingFace bezogen.

7 Evaluation

In diesem Kapitel werden die Experimente umfangreich evaluiert. Dies geschieht einerseits automatisiert auf Basis der ausgewählten Metriken, andererseits manuell in Form einer qualitativen Analyse der aus den Anhängen A und B generierten Zusammenfassungen. Die Experimente werden zwecks anschließender Zuordnung wie folgt kodiert:

ID	**Experiment**
1	Reproduktion auf englischen Daten
2	Adaption auf deutschen Daten
3	Adaption auf multilingualen Daten

Tabelle 7.1: Kodierung der Experimente.

7.1 Automatische Auswertung

In Tabelle 7.2 sind die ROUGE-Scores der TF2TF aller Experimente gesamtheitlich dokumentiert. Dabei ist die Konfiguration der Experimente jeweils anhand der ID, der genutzten DLR und der Sprache der eingegangenen Textdaten nachvollziehbar. Die drei letztgenannten Spalten sind als zusammengesetzter Primärschlüssel zu verstehen.

ID	**DLR**	**Sprache**	**R-Recall**	**R-Precision**	**R-Measure**
1	BERT	EN	15.78	10.25	12.11
1	XLM-R	EN	16.49	17.12	16.25
2	BERT	DE	15.93	10.52	11.97
2	XLM-R	DE	15.93	10.32	11.86
3	BERT	EN & DE	16.76	11.18	12.68
3	XLM-R	EN & DE	16.44	15.45	15.35
3	BART	EN & DE	15.01	12.43	13.10

Tabelle 7.2: Ergebnisse im Experimentvergleich.

Experiment 1: Reproduktion auf englischen Daten
Im ersten Experiment geschah die Reproduktion des SOTA auf Basis englischsprachiger Textdaten. Das TF2TF ist demnach unter Nutzung von BERT und XLM-R weitestgehend SOTA-konform. Im Bereich der XLM-R ist jedoch ein ungewöhnlich hoher R-Precision-Score auffallend, welcher sich in der Berechnung auch auf den R-Measure-Score auswirkt. Dies bedeutet per Formel eine hohe Übereinstimmung zwischen den Wörtern der Originaltexte und der generierten Zusammenfassungen. Eine weitergehende Ursache hierfür wird im nachfolgenden Kapitel ergründet. Die Lernfortschritte sind anhand der Fehlerentwicklung in Abbildung 7.1 visualisiert. Hierbei ist zu erkennen, dass XLM-R erst spät, dann aber ebenbürtig lernt.

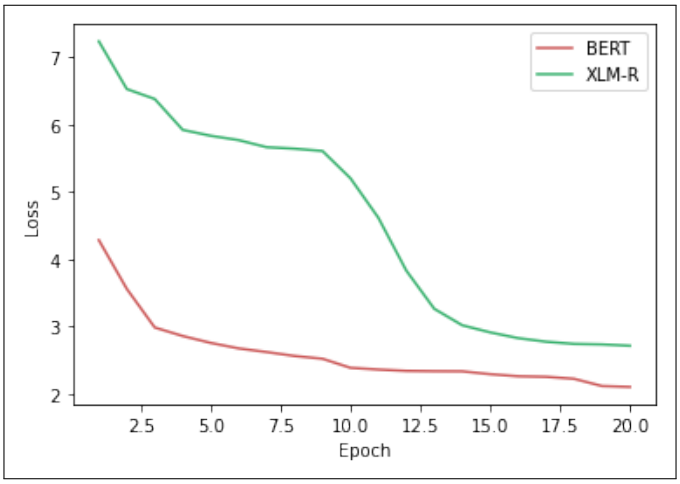

Abbildung 7.1: Loss in der SOTA-Reproduktion im Modellvergleich.

Experiment 2: Adaption auf deutschen Daten
Im zweiten Experiment geschah erstmalig die sprachtechnische Adaption der TF2TF. Hierfür kamen demnach ausschließlich deutschsprachige Daten zum Einsatz. Tabelle 7.2 gibt hierbei zu erkennen, dass das Modell unter Nutzung von BERT qualitativ in Englisch genauso gut wie in Deutsch ist. Im Bereich der XLM-R sind nun keine auffallenden Werte mehr zu verzeichnen. Wertmäßig gleichen sich die TF2TF nun unter Nutzung beider DLR unter Akzeptanz eines gewissen Rauschens auf SOTA-Niveau. Die Lernfortschritte sind in bekannter Weise in Abbildung 7.2 visualisiert.

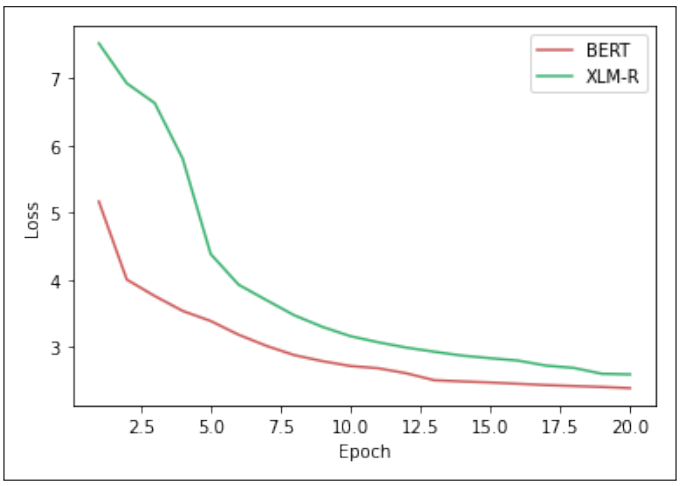

Abbildung 7.2: Loss in der Adaption auf deutschen Daten im Modellvergleich.

Experiment 3: Adaption auf multilingualen Daten

Im dritten Experiment geschah die sprachtechnische Adaption der TF2TF auf Basis multilingualer Textdaten. BART wird indes aus externer Quelle bezogen. Die Evaluation geschieht im Kontext dieser Arbeit ausschließlich auf deutschsprachigen Textdaten. In Tabelle 7.2 wird ersichtlich, dass sich das Modell unter Nutzung von BERT nun über den SOTA hinaus verbessert, während im Bereich der XLM-R erneut auffallende Werte entstehen. BART entspricht ebenfalls dem SOTA. Hier sei jedoch angemerkt, dass er den SOTA auf Basis einer englischsprachigen Evaluation übersteigt. In Abbildung 7.3 ist abermals die recht späte Konvergenz der XLM-R nachvollziehbar, welche womöglich auf die architektonischen Unterschiede der DLR zurückzuführen ist.

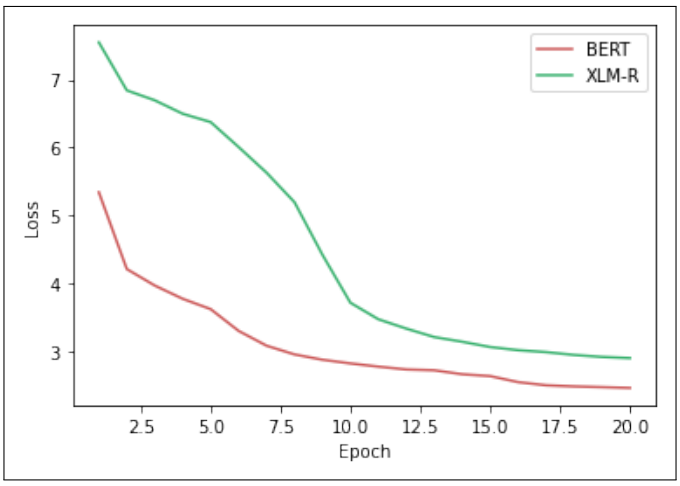

Abbildung 7.3: Loss in der Adaption auf multilingualen Daten im Modellvergleich.

7.2 Qualitative Analyse

Die soeben wertmäßig evaluierten Experimente werden nun ergänzend qualitativ analysiert. Hierfür werden die TF2TF aller Experimente sprachabhängig entweder Anhang A oder Anhang B unterzogen, indem die dort enthaltenen mitunter herausfordernden Texte maschinell zusammengefasst und subjektiv bewertet werden. Die Bewertung geschieht anhand des ursprünglichen Textes und dem angefügten Gold-Standard. Eine derartige Analyse ist erforderlich, um die Qualität sowie die praktische Eignung der trainierten TF2TF vollends beurteilen zu können. Die qualitative Analyse ist außerdem erforderlich, um den teilweise kritisch begutachteten sowie statistisch veranlagten ROUGE-Score qualitativ einzuordnen. Die in diesem Kapitel referenzierten Zusammenfassungen sind in englischer und deutscher Sprache modellübergreifend stets den Anhängen A und B zu entnehmen.

Experiment 1: Reproduktion auf englischen Daten
Ergänzend zum ersten Experiment ist nun zu analysieren, ob das TF2TF unter Nutzung von BERT tatsächlich SOTA-konform ist und wie die auffallenden Werte unter Nutzung von XLM-R zu erklären sind.

BERT scheint die beispielhaften Texte sehr gut verstehen zu können. Dies war zu erwarten, da BERT bekanntermaßen hervorragend als Encoder funktioniert, etwas beschwerlicher hingegen als Decoder. Dies ist jedoch bis auf eine Ausnahme im Text über das Finale der WM 2014 nicht herauszulesen. Dabei sind die entsprechenden Zusammenfassungen nicht zur orthographisch und grammatikalisch korrekt, sondern weitestgehend auch mit den wichtigsten Informationen ausgestattet. Das TF2TF unter Nutzung von BERT gilt daher tatsächlich als SOTA-konform.

XLM-R verspricht eine wertmäßige Verbesserung. In den generierten Zusammenfassungen wird die inhaltliche Unzulänglichkeit jedoch für den menschlichen Leser auch ohne Kenntnis über die ursprünglichen Texte sehr schnell deutlich. Gleichermaßen ist hier nachvollziehbar, wie die hohen Werte im Bereich der R-Precision und der R-Measure entstanden sind: Die als wichtig erkannten Worte oder Wortpassagen der ursprünglichen Texte werden in den Zusammenfassungen sehr häufig wiederholt. Diese Form der Exploitation hat im Kontext der ATS einen höchst nachteiligen Effekt, da inhaltlich keine qualitativen Zusammenfassungen entstehen. Die gegenüber BERT komplexere Architektur der XLM-R scheint mit den zugeführten Textdaten nicht korrekt lernen zu können und gilt hiermit nicht mehr als SOTA-konform.

Experiment 2: Adaption auf deutschen Daten
Ergänzend zum zweiten Experiment ist zu analysieren, ob die Adaption der TF2TF unter Nutzung von BERT und XLM-R auf deutschen Daten qualitativ ist. BERT scheint die beispielhaften Texte trotz gleichbleibender Werte mit vermehrt fehlerhaften sowie teils widersprüchlichen Informationen zusammenzufassen. Dies ist auch gleichbleibend, wenn BERT durch German BERT (GBERT) ausgetauscht wird. XLM-R weist trotz normalisierter Werte auf SOTA-Niveau ein ähnlich fehlerbehaftetes Verhalten auf.

Hierbei wird einerseits die Abhängigkeit der Modelle vom Umfang der Textdaten deutlich. Dies betrifft sowohl das Vortraining der verwendeten DLR als auch das Fine-Tuning der TF2TF. Dabei stehen in englischer Sprache so viele Textdaten wie in keiner anderen Sprache frei zur Verfügung. Folglich ist es nicht verwunderlich, wenn die verborgenen Strukturen anderer Sprachen nicht vollends erlernt werden können und die Qualität dadurch limitiert wird.

Andererseits wird die Abhängigkeit der Modelle von der Beschaffenheit der Textdaten deutlich, insbesondere anhand der Natur der Zusammenfassungen. Hier ist nämlich ein typisches Element der Nachrichtendomäne erkennbar, da anstatt einer konventionellen Zusammenfassung oftmals ein Teaser generiert wird, der den Leser binden und zum eigentlichen Text weiterleiten soll. Dieses Verhalten ist jedoch nicht auf das Modell zurückzuführen, sondern auf die Trainingsdaten, welche größtenteils eben jener Nachrichtendomäne entstammen. Dies unterstreicht insgesamt die Notwendigkeit einer intensiven Datenanalyse. Hinsichtlich eines praktischen Einsatzes ist also stets ein anforderungsgerechter Korpus zu formieren. Zudem ist der Einfluss des vortrainierten Modells bemerkbar, beispielsweise im Text über das Finale der WM 2014. Hier wird im originalen Text von Jogi Löw geschrieben, in der Zusammenfassung hingegen von Jürgen Klopp. Letzterer wurde zuvor jedoch nie erwähnt. Dies kann mit der geringen Distanz beider Personen im Vektorraum erklärt werden und ist somit zwar sachlogisch falsch, mathematisch jedoch nachvollziehbar. Dies tritt in ähnlicher Weise auch in den anderen Zusammenfassungen auf. Ein Fine-Tuning mit sehr viel mehr Trainingsdaten verspricht dieses Rauschen hinreichend zu minimieren.

Experiment 3: Adaption auf multilingualen Daten
Ergänzend zum dritten Experiment ist zu analysieren, inwiefern wie wertmäßigen Fortschritte auch in den generierten Zusammenfassungen erkennbar sind. Außerdem ist erstmalig die Qualität von BART zu untersuchen.

Das TF2TF unter Nutzung von BERT verbessert sich in Englisch nun bis hin zur praktischen Eignung, in Deutsch jedoch nicht. Das TF2TF unter Nutzung von XLM-R erzielt hingegen keine Verbesserung. Dies ist vermutlich auf die erneut ungewöhnlich hohen Werte im Bereich der R-Precision und der R-Measure zurückzuführen. XLM-R scheint große Probleme mit den vorliegenden englischsprachigen Textdaten zu haben.

BART weist aufgrund der identischen Datengrundlage ein ähnliches Verhalten wie die monolingualen TF2TF auf, scheint jedoch insgesamt robuster zu sein und weniger Fehler zu machen. Dennoch sind auch hier inhaltliche Mängel sowie erstmalig auch englische Wörter in deutschen Zusammenfassungen zu verzeichnen. Entgegen obig formulierter Erwartung scheint BART dem lockenden Element der Nachrichtendomäne nicht nachzukommen. BART ist für die ATS in englischer Sprache das beste Modell, in deutscher Sprache vermutlich nur mit einem hinreichenden Fine-Tuning. BERT ist in englischer Sprache wertmäßig und subjektiv wider Erwarten nur leicht unterlegen, in deutscher Sprache hingegen deutlich.

Evaluation der Gold-Standards
Vergleicht man die Gold-Standards mit den maschinell generierten Zusammenfassungen, so werden die oben beschriebenen Eindrücke bestätigt. Tabelle 7.3 vergleicht die ROUGE-Scores aller multilingual vortrainierten Modelle auf Basis eben dieser Gold-Standards.

DLR	**Sprache**	**R-Recall**	**R-Precision**	**R-Measure**
BERT	EN	24.41	25.12	24.58
XLM-R	EN	18.35	19.19	18.55
BART	EN	26.12	20.54	22.91
BERT	DE	6.98	10.74	8.41
XLM-R	DE	10.20	12.84	11.16
BART	DE	11.49	16.01	12.94

Tabelle 7.3: ROUGE-Scores der multilingual vortrainierten Modelle auf Basis der Gold-Standards.

Inwieweit diese Werte belastbar sind, ist außerhalb dieser Arbeit zu ergründen. Es lässt sich nur mutmaßen, warum die Werte derart unterschiedlich ausfallen: Menschen können beim Verfassen der Gold-Standards beispielsweise Kontextwissen nutzen, welches gar nicht im ursprünglichen Text stand. Zudem besteht die Hypothese, dass Menschen unterbewusst N-Gramme aus dem ursprünglichen Text übernehmen, wodurch die Werte ungewöhnlich hoch sind. Abschließend ließe sich beurteilen, ob die in Tabelle 7.3 dokumentierten Werte als obere Grenze interpretiert werden können.

8 Fazit

Die bereits evaluierten und analysierten Ergebnisse werden nun zusammengefasst und weitergehend diskutiert. Dabei wird neben der Zusammenfassung insbesondere das Vorgehen reflektiert, die praktischen Implikationen dargestellt und weiterer Forschungsbedarf abgeleitet.

8.1 Zusammenfassung

In dieser Arbeit wurde die Adaption multilingual vortrainierter Modelle zur automatischen Zusammenfassung von Texten auf die deutsche Sprache erforscht und demonstriert. Hierfür wurden entsprechende Grundlagen in DL und NLP dargelegt. Dabei ist insbesondere der kontextbezogene Fortschritt durch TL in Verbindung mit den eingeführten DLR hervorzuheben. Um das Ziel dieser Arbeit zu erreichen und die Forschungsfragen hinreichend zu beantworten, schlossen sich verschiedene Experimente an, deren Architektur und Datengrundlage zuvor methodisch aufbauend definiert wurde.

Das Ziel dieser Arbeit, multilingual vortrainierte Modelle mittels TL auf die deutsche Sprache zu adaptieren, konnte mithilfe von TF2TF unter Nutzung von BERT und BART erreicht werden. Die entsprechenden Architekturen erreichen den SOTA der ATS in Englisch sowie in Deutsch. Die analysierten qualitativen Schwächen sind jeweils mit existierenden SOTA-Modellen vergleichbar und daher mit den genannten Einschränkungen akzeptabel. Hierbei ist aus wissenschaftlichem Interesse außerdem anzumerken, dass mit HuggingFace selbst eines der größten Unternehmen der Branche, welches theoretisch über entsprechende Ressourcen verfügt, keine optimalen Ergebnisse erzielt. Es folgt die Beantwortung der einleitend formulierten Forschungsfragen.

Wie lassen sich Texte automatisiert zusammenfassen?
Hierfür bedarf es einer Architektur, welche die Grundlagen des NLP vereint, entsprechend vortrainierte DLR integriert und ein weitergehendes Fine-Tuning auf möglichst vielen paarweisen Textdaten ermöglicht, um die Herausforderungen des NLU und der NLG zu bewältigen. Dies wird beispielsweise von einem Sequence-to-Sequence-Transformer-Modell, welches über einen Encoder und einen Decoder verfügt, erfüllt.

Wie können existierende Modelle auf eine andere Sprache adaptiert werden?
Hierfür sind nicht zwingend architektonische Anpassungen notwendig, sondern vielmehr ein Austausch der Textdaten in der entsprechenden Zielsprache. Die oben beschriebene Architektur ist folglich in der Lage, sprachabhängige Strukturen dynamisch zu erlernen. Dabei wird die Qualität der Adaption entscheidend von den vortrainierten Modellen sowie den zugeführten Textdaten beeinflusst.

Wie qualitativ und skalierbar ist die Lösung?
Während die untersuchten Architekturen wertmäßig den SOTA erreichen, konnten qualitativ verschiedene Eigenarten identifiziert werden. Dies ist jedoch meist nicht den Architekturen selbst anzurechnen, sondern den zugeführten Textdaten, da sich die Architekturen in anderen Sprachen mit entsprechend umfangreichen und qualitativen Textdaten bereits bewährt haben. Daher ist gleichermaßen nicht davon auszugehen, dass die Architekturen größere Probleme mit der deutschen Sprache haben. Wurde ein Modell erst einmal hinreichend trainiert, kann es veröffentlicht, in Anwendungen implementiert und somit entsprechend skaliert werden. Hierbei ist das Echtzeitverhalten jedoch hinreichend zu analysieren, insofern dies im Anwendungsfall relevant ist. Eine solche Skalierung ist Menschen nicht möglich.

8.2 Reflexion des Vorgehens

Das Vorgehen in dieser Arbeit hat sich insgesamt bewährt. Dies bestätigt die Zielerreichung und die Beantwortung der Forschungsfragen. Nichtsdestotrotz gab es Abweichungen vom ursprünglichen Plan. Neben geringfügigen Änderungen am Aufbau und den Inhalten der Arbeit ist hiermit vornehmlich die zeitliche Verzögerung aufgrund der Abhängigkeit von Dritten gemeint.

Die Suche, Auswahl und Formation einer geeigneten deutschsprachigen Datengrundlage gestaltete sich komplexer als vermutet und erwies sich überdies als sehr zeitintensiv. Zudem nahm die Einrichtung des Hochleistungsrechners einige Wochen in Anspruch, unter anderem wegen systemseitigen technischen Problemen und verzögerten Antworten bei der Kontaktaufnahme.

8.3 Praktische Implikationen

Es ist zu erkennen, dass die trainierten Modelle datenentsprechend lernen. Die Lernfortschritte in Englisch und nach erfolgter Adaption auch in Deutsch bestätigen die Eignung der TF2TF für die ATS, insbesondere unter Nutzung von BERT und BART. Hier sei auf die Gold-Standards in den Anhängen A und B verwiesen. Dabei können die Herausforderungen des NLU und der NLG bewältigt werden. Praktisch ist die erprobte Architektur somit aus technischer und qualitativer Sicht implementierfähig, insofern ein domänen- und sprachspezifisches Training durchgeführt wird. Die entsprechenden Anforderungen, unter anderem an die erforderliche Datengrundlage, werden nachfolgend im Forschungsbedarf offengelegt. Ergänzend bedarf es zwingend einer nachgelagerten Anwendung, welche das entsprechend vortrainierte Modell einbindet. Das Modell wird in einem gewöhnlichen Format abgespeichert, sodass die Integration problemlos erfolgen kann.

8.4 Weiterer Forschungsbedarf

Um die Qualität der Adaption vor allem im deutschsprachigen Raum hinreichend zu verbessern, bedarf es unbedingt einer überarbeiteten Datengrundlage. Hat diese keinen angemessenen Umfang, so sind die trainierten Modelle gemäß Overfitting überparametrisiert. Dies führt mitunter dazu, dass Wörter, welche je nach Domäne in unterschiedlichen Kontexten vorkommen, falsch verstanden werden. Außerdem entstehen Probleme bei Wörtern, die den Wortschatz der trainierten Modelle übersteigen. Damit eben jene Modelle besser generalisieren, ist also zunächst der Umfang der Datengrundlage deutlich zu erweitern. Dabei sind stets die bekannten Anforderungen zu erfüllen. Aufgrund des qualitativen Einflusses sind hierbei unbedingt die Eigenarten der verwendeten Textdaten zu analysieren und entsprechend vorzuverarbeiten. Hier bietet es sich an, Textdaten der Zieldomäne zu nutzen, um die Qualität der ATS im Kontext des jeweiligen Anwendungsfalls zu erhöhen. Eine allgemeinsprachliche Nutzung bedarf entsprechend noch umfangreicherer Textdaten. Den Umfang der Datengrundlage gilt es also stets zu maxi-

mieren, während die Eigenarten der Textdaten dem Anwendungsfall gerecht sein sollten. Zudem ist festzuhalten, dass sich die Sprache und die Domäne der ATS über die im jeweils erforderlichen Fine-Tuning zugeführten Textdaten steuern lässt.

Möglich sind außerdem Ansätze wie die sogenannte Data Augmentation oder auch die Übersetzung anderssprachiger Texte, um künstlich neue Textdaten für den deutschsprachigen Raum zu erzeugen. Hier ist die entstehende Qualität der Textdaten in jedem Fall mit höchster Vorsicht zu untersuchen. Um also weitergehende Fortschritte in der deutschsprachigen ATS zu erzielen, sind verschiedenartige Projekte, welche sich ausschließlich auf die Formation neuer Korpora konzentrieren, unabdingbar.

Das Pareto-Prinzip kann indes vom Menschen auf die Maschine übertragen werden und knüpft hier unmittelbar an. Demnach erfolgen 80% der Arbeit in nur 20% der Zeit. Die letzten 20% der Arbeit bedürfen folglich ganze 80% der Zeit. Hier passieren allerdings ganz entscheidende Dinge. Dies bedeutet, dass das TF2TF mit fortwährender Trainingsdauer etliche Feinheiten der zugeführten Textdaten erlernt, welche sich erheblich auf die Qualität auswirken. Gleichzeitig lassen sich entsprechende Schwächen der generierten Zusammenfassungen auf eben diese fehlende Phase zurückführen, welche aus unzureichenden Textdaten resultiert. Eine umfangreichere Datengrundlage, bestenfalls mit einem Umfang im Millionenbereich, ist daher umso wichtiger.

Darüber hinaus ist es je nach Anwendungsfall beispielsweise sinnvoll, das Verhalten der ATS unter Zugabe kurzer und langer Texte zu analysieren. Anpassungen lassen sich bei Bedarf mit einem sogenannten Sliding Window Approach, welcher das methodische Teilen und Konkatenieren der Texte vorsieht, vornehmen. Hier sei angemerkt, dass HuggingFace bereits entsprechende Methoden mitliefert und daher immerhin strukturbezogene Exploitation vorbeugt.

Im Verlauf der Experimente fielen zudem wiederholt ungewöhnliche ROUGE-Scores auf. Zumeist entstammt auch dieses Phänomen den Eigenarten der zugrundeliegenden Textdaten. Dennoch sind geeignete Metriken zu erforschen und auszuwählen, welche es im Kontext der ATS ermöglichen, das Training vor einem potenziellen Overfitting zu stoppen. Dies würde jedoch folglich eine äußerst komplizierte Messung des Generalisierungseffektes bedeuten und ist daher eine wesentliche Herausforderung, deren Bewältigung einen ähnlich großen Durchbruch in der Wissenschaft der ATS bewirken könnte, wie es die vortrainierten DLR taten.

Weiterhin ist die Optimierung der Hyperparameter notwendig. Zwar ist dies bei extern vortrainierten Modellen wie den DLR nur eingeschränkt möglich, dafür bietet TF2TF entsprechende Konfigurationsmöglichkeiten. Die Hyperparameter sind entweder vor oder adaptiv während des Trainings zu definieren. Meist sind die Ergebnisse bereits nach der Zugabe von 10-20% der eigentlichen Daten repräsentativ und somit vergleichbar, weshalb die Optimierung keine unzähligen vollständigen Trainingsläufe erfordert. Neben der LR und der Batch Size ist unter anderem auch die Anzahl der Aufwärmschritte, die Anzahl der verborgenen Schichten und die Anzahl der Attention-Heads zu bestimmen. Darüber hinaus können verschiedene Parameter des verwendeten Tokenizers konfiguriert werden. Nicht zuletzt ist das Teilen der Gewichte zwischen dem Encoder und dem Decoder zu erproben. In der verwendeten Konfiguration des TF2TF werden Texte mit einer Länge von 512 Token akzeptiert, während entsprechende Zusammenfassungen auf 128 Token limitiert sind. Dies entspricht einer Kompressionsrate von 75%. Die Natur der entstehenden ATS ist nicht nur über diese Werte, sondern abermals über die zugeführten Textdaten bestimmbar. Somit lässt sich erwartungsgemäß je nach Anwendungsfall auch die Kompressionsrate bedarfsgerecht steuern [Nitsche, 2019, S. 14-15].

In Bezugnahme auf die zu Beginn konstruierten Einsatzgebiete aus dem Gesundheitswesen lassen sich folgende Maßnahmen ableiten: Die Datengrundlage ist mit umfangreichen medizinischen Texten zu erweitern, bevor ein entsprechendes Fine-Tuning mit anschließender Optimierung der Hyperparameter erfolgen kann. Es existieren hierbei jedoch neue domänenspezifische Gefahren. Demnach gestaltet sich insbesondere die Zusammenfassung von Patientengesprächen als höchst kompliziert, weil initial bereits eine hohe Informationsdichte vorliegt und somit wahrscheinlich ein gewisser Informationsverlust entsteht.

Unter entsprechenden Voraussetzungen ist es außerdem möglich, das Vortraining einer ausgewählten oder selbst konzipierten Architektur selbst zu übernehmen. Somit ließe sich die maximal einzubettende Textlänge steuern. Der hierbei entstehende Rechenbedarf ist jedoch nicht zu unterschätzen.

A Qualitative Analyse (Englisch)

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.“

Zusammenfassungen von Text 1

Gold-Standard:

The perpetrators destroyed symbolic buildings, the White House and the Pentagon. 2750 people died in the WTC, 147 in airplanes and 184 in the Pentagon. The terrorist attack was the first military attack on the U.S. since 1814. President Bush announced that he would punish the perpetrators.

BERT (monolingual):

The perpetrators knew what they were doing. The White House and the Pentagon as symbols of power, the World Trade Center as a symbol of capital and business. The attacks were media-ready and media-prepared. The perimeters were precisely chosen: The White House.

BERT (multilingual):

The attacks were media-ready, but the perpetrators knew what they were doing. The attacks are a symbol of capital and business, the Pentagon says. The U.S. are investigating the attacks, the White House says.

XLM-R (monolingual):

The attacks were media-ready, the Pentagon and the Pentagon. The attack was taken by the Pentagon as a symbol of power, the World Trade Center. The perpetrators were used by power.

XLM-R (multilingual):

In the wake of the World Trade Center attacks, the U.S. has seen a huge surge in violence. The attacks are part of a global global economy. It's not the first time that the attacks occur in the United States. In the past two years, the world has seen an uncertainty.

BART (multilingual):

U.S. proved vulnerable at home on Sept. 11, 2001, for the first time since independence struggles. 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 symbol of capital and business.

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.

Zusammenfassungen von Text 2

Gold-Standard:

Joachim Löw did not change his starting lineup after the 7-1 win against Brazil. Shortly before the game, he had to replace Khedira with Kramer. Higuain scored for Argentina but was offside. The teams took fewer and fewer risks. Two players from Argentina were lucky not to get a sending off. Götze decided the game with a dream goal.

BERT (monolingual):

Joachim Löw saw no reason to change his starting eleven. Khedira had to pull out at short notice with calf problems. He was replaced in the starting line-up by Kramer. Kramer was replaced by Kramer in the first half. The DFB team tried hard but the South Americans were more determined.

BERT (multilingual):

Joachim Löw saw no reason to change his starting eleven after 7-1 win over Brazil. The DFB team tried hard to change their starting eleven. Khedira had to pull out at short notice with calf problems. He was replaced in the starting lineup by the South Americans.

XLM-R (monolingual):

Khedira had to pull out at short notice with calf problems. Kramer replaced the coach Joachim Löw. The coach was replaced in the starting lineup. Joachim Löw replaced his starting eleven 7-1 victory against host Brazil.

XLM-R (multilingual):

Joachim Löw hat die 7-1 triumph in der 7-1-Niederlage gegen Gastgeber Brazil gelassen. Joachim Löws 7-1 wins in der Champions League gegen Brazil. Joachim Gauck scheitert an der WM-Qualifikation in Brasilien.

BART (multilingual):

Joachim Löw saw no reason to change his starting eleven after Germany's 7-1 win over Brazil. Mascherano and Aguero were lucky not to be sent off early with a yellow card. The tempo of the match dwindled with the courage of the South Americans. Götze opened the scoring for Germany with a left-footed volley.

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.

Zusammenfassungen von Text 3

Gold-Standard:

Inflation is a sustained process in which the purchasing power of money is permanently reduced. It is measured by the inflation rate. Prices rise, wages rise, and thus demand also rises. Higher wages in turn cause higher costs for companies, which is why prices for goods continue to rise.

BERT (monolingual):

Inflation manifests itself in general price increases. Inflation does not include one-time, temporary price level increases caused by unusual occurrences or price increases for specific goods or production factors. The purchasing power of money is permanently reduced.

BERT (multilingual):

Inflation is a sustained process of demonetization, which manifests itself in general price increases. Inflation does not include one-time, temporary price level increases caused by unusual occurrences or price increases for specific goods or production factors. The purchasing power of money is permanently reduced.

XLM-R (monolingual):

The purchasing power is a sustained process of demonetization, which manifests itself in general price increases. A sustained power of money is permanently reduced by unusual occurrences or price increase.

XLM-R (multilingual):

Inflation is a sustained process of demonetization. Inflation does not include one-time, temporary price level increases. The purchasing power of money is permanently reduced. In general price increases are permanently decreased by one factor.

BART (multilingual):

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.

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.

Zusammenfassungen von Text 4

Gold-Standard:

If a traveler cancels the trip due to a virus pandemic, the tour operator is not entitled to compensation. Travelers therefore do not have to pay a cancellation fee if the tour operator has also canceled the trip.

BERT (monolingual):

Tour operator is not entitled to compensation under Section 651 HGB. 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 is the result of a virus pandemic in the German Civil Code.

BERT (multilingual):

If a traveler cancels the trip due to a virus pandemic, the tour operator is not entitled to compensation. The lack of treatment options and vaccination justifies a right to cancel the trip. The district court Stuttgart has ruled that there is no treatment for the virus.

XLM-R (monolingual):

Tour operator cancels the trip due to a virus pandemic. The tour operator is not entitled to compensation. In addition, the lack of treatment options and vaccination justifies a right to cancel the trip free of charge.

XLM-R (multilingual):

If a traveler cancels the trip due to a virus pandemic, the tour operator is not entitled to compensation. If a trip is canceled, the trip operator is entitled to refund the trip free of charge. The court Stuttgart has decided the trip will be free of charges.

BART (multilingual):

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. In addition, the lack of treatment options and vaccination justifies a right to cancel the trip free of charge. The Stuttgart District Court ruled in favor of the traveler.

Text 5: Anleitung zum Händewaschen

Washing your hands is easy, and it's one of the most effective ways to prevent the spread of germs. Clean hands can stop germs from spreading from one person to another and throughout an entire community. Follow these five steps every time. Wet your hands with clean, running water (warm or cold), turn off the tap, and apply soap. Lather your hands by rubbing them together with the soap. Lather the backs of your hands, between your fingers, and under your nails. Scrub your hands for at least 20 seconds. Rinse your hands well under clean, running water. Dry your hands using a clean towel or air dry them.

Zusammenfassungen von Text 5

Gold-Standard:

Hand washing is the most effective way to prevent the spread of germs. For this, you should wash your hands with cold water, then add soap, rub properly for 20 seconds and then rinse under cold water.

BERT (monolingual):

Clean hands can stop germs spreading from one person to another. Clean hands with clean, running water, and apply soap. Lather your hands by rubbing them together with soap. Scrub your hands for at least 20 seconds. Lather the backs of your hands between your fingers and under your nails.

BERT (multilingual):

Washing your hands is easy, and it's one of the most effective ways to prevent the spread of germs. Clean hands can stop germs from spreading from one person to another.

XLM-R (monolingual):

Clean hands can stop germs from one person to another. Clean hands with clean, running water (warm or cold), turn off the tap, and apply soap. Follow the five steps every time, you're one of the most effective ways to prevent the spread of germs.

XLM-R (multilingual):

Clean hands can stop germs from spreading from one person to another. Clean hands for at least 20 seconds every time you use the soap. Follow these five steps every time.

BART (multilingual):

Washing your hands is one of the most effective ways to prevent the spread of germs. Clean hands can stop germs from spreading from one person to another and throughout an entire community. Wet your hands with clean, running water (warm or cold), turn off the tap, and apply soap.

B Qualitative Analyse (Deutsch)

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.“

Zusammenfassungen von Text 1

Gold-Standard:

Die Täter zerstörten symbolische Gebäude, das weiße Haus und das Pentagon. 2750 Menschen starben im WTC, 147 in Flugzeugen und 184 im Pentagon. Der Terroranschlag war der erste militärische Angriff auf die USA seit 1814. Präsident Bush kündigte an, dass die Täter gejagt und bestraft werden.

BERT (monolingual):

Die Anschläge in New York und Washington zeigen, wie die Terroristen in den USA die Macht verloren haben. Eine Übersicht über die Attentäter, die die Terrormiliz bekämpfen will.

BERT (multilingual):

Die Täter wussten, was sie taten: Was die Täter in den USA taten, war Zerstörung. Und wie sie die Stadt und das Pentagon als Symbole der Macht bewerten. Die Bilder der Anschläge in Berlin.

XLM-R (monolingual):

Die Anschläge auf die World Trade Center waren die ersten Anschläge in den USA. Sie waren die letzten, in denen sich die USA und die USA aufeinander einlassen mussten.

XLM-R (multilingual):

In den USA ist die Demokratie ein Symbol des Kapitalismus geworden. Die Bilder von der Revolution sind in den USA immer noch präzise geworden. Doch die Bilder sind längst nicht mehr so schön, wie es ist.

BART (multilingual):

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. Die Anschläge waren der erste quasi militärische Angriff auf die Vereinigten Staaten seit 1814.

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.

Zusammenfassungen von Text 2

Gold-Standard:

Joachim Löw änderte seine Startaufstellung nach dem 7:1-Sieg gegen Brasilien nicht. Kurz vor dem Spiel musste er Khedira durch Kramer ersetzen. Higuain erzielte ein Abseitstor für Argentinien. Die Mannschaften gingen im Spielverlauf immer weniger Risiko ein. Zwei argentinische Spieler hatten Glück, dass sie keinen Platzverweis bekommen haben. Götze entschied das Spiel mit einem Traumtor auf Vorlage von Schürrle.

BERT (monolingual):

Die DFB-Elf verliert im Halbfinale gegen Brasilien mit 7:1. Das Team von Joachim Löw zeigt, wie sehr sich das Team von Jürgen Klopp in der Startelf entwickelt hat - und warum es nicht so gut ist, sich in die WM zu locken.

BERT (multilingual):

Die DFB-Elf verliert das Halbfinale gegen Brasilien mit 7:1. Der Sieg ist nicht das erste Mal, dass die Südamerikaner den Sieg errungen haben - und die DFB-Frauen sind nicht zufrieden mit dem Spiel.

XLM-R (monolingual):

Bundestrainer Joachim Löw gewinnt das Halbfinale gegen Gastgeber Brasilien mit 7:1. Die DFB-Elf holt sich den ersten Sieg in Serie, die deutsche Mannschaft holt den ersten Saisonsieg in Serie.

XLM-R (multilingual):

Der DFB-Elf gewinnt gegen Brasilien und holt sich den Sieg gegen Brasilien. Der FC Bayern beim 7:1-Erfolg gegen Gastgeber Brasilien. Die Bayern beim 0:1 in der 2. Liga in der Einzelkritik.

BART (multilingual):

Joachim Löw beat Brasilien 7:1 im Halbfinale gegen Gastgeber Brazil 7-1 in Berlin. Khedira meldete sich with Wadenproblemen kurzfristig ab, für ihn rückte Kramer in die Startformation.

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.

Zusammenfassungen von Text 3

Gold-Standard:

Inflation ist ein anhaltender Prozess, bei dem die Kaufkraft des Geldes dauerhaft verringert wird. Sie wird durch die Inflationsrate gemessen. Die Preise steigen, die Löhne steigen, und damit steigt auch die Nachfrage. Höhere Löhne verursachen wiederum höhere Kosten für Unternehmen, weshalb die Preise für Güter weiter steigen.

BERT (monolingual):

Die Inflation ist ein anhaltender Prozess der Geldentwertung, der sich durch allgemeine Preiserhöhungen bemerkbar macht. Doch es gibt auch eine Lösung: Die Kaufkraft des Geldes verringert sich, die Kaufpreise steigen.

BERT (multilingual):

Die Inflation ist ein anhaltender Prozess der Geldentwertung, der sich durch allgemeine Preiserhöhungen bemerkbar macht. Doch die Kaufkraft des Geldes verringert sich, weil die Kaufsumme immer höher ist als gedacht.

XLM-R (monolingual):

Die Inflation ist ein anhaltender Prozess der Geldentwertung, der sich durch allgemeine Preiserhöhungen bemerkbar macht. Der Prozess wird von der Inflation erfasst und wird von einer Geldeinheit geführt. Die Preise für Güter und Produktionsfaktoren werden von der gegenwärtigen Inflation abgeleitet.

XLM-R (multilingual):

Die Inflation ist ein anhaltender Prozess der Geldentwertung, der sich durch allgemeine Preiserhöhungen bemerkbar macht. Die Preise für bestimmte Güter werden immer teurer. Die Inflation ist ein Anlaufmodell für die steigenden Preise.

BART (multilingual):

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.

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.

Zusammenfassungen von Text 4

Gold-Standard:

Wenn ein Reisender die Reise aufgrund einer Viruspandemie storniert, hat der Reiseveranstalter keinen Anspruch auf Entschädigung. Reisende müssen daher keine Stornogebühren zahlen, wenn auch der Reiseveranstalter die Reise storniert hat.

BERT (monolingual):

Ein Reiseveranstalter muss die Reise absagen, wenn er die Reise nicht absagt. Das entschied das Bundesamt für Verkehrswesen. Das Urteil ist nicht nur für Reisende, sondern auch für das Reiserücktrittsrecht.

BERT (multilingual):

Ein Reisender wegen einer Virus-Pandemie muss sich deswegen nicht entschädigen. Das Amtsgericht Stuttgart hat entschieden, dass die fehlende Therapiemöglichkeit und Impfung ein kostenloses Reiserücktrittsrecht geltend machen.

XLM-R (monolingual):

Das Amtsgericht Stuttgart hat entschieden, dass ein Reiseveranstalter die Reise absagen muss, wenn er wegen einer Virus-Pandemie die Reise abgesagt hat. Das hat das Amt nun entschieden. Das Urteil könnte auch für Reisende gelten. Ein Überblick über die wichtigsten Punkte.

XLM-R (multilingual):

Das Amtsgericht Stuttgart hat entschieden, dass ein Reiseveranstalter wegen einer Virus-Pandemie die Reise absagen muss, wenn er sich nicht auf Entschädigungen einsetzt.

BART (multilingual):

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. Zudem begründet die fehlende Therapiemöglichkeit and Impfung ein kostenloses Reiserücktrittsrecht.

Text 5: Anleitung zum Händewaschen

Händewaschen ist eine der effektivsten Möglichkeiten, die Verbreitung von Keimen zu verhindern. Saubere Hände können die Ausbreitung von Keimen zwischen Personen und innerhalb einer ganzen Gemeinschaft verhindern. Befolgen Sie stets diese fünf Schritte. Befeuchten Sie Ihre Hände mit sauberem, fließendem Wasser (warm oder kalt), drehen Sie den Wasserhahn zu und verwenden Sie Seife. Schäumen Sie Ihre Hände ein, indem Sie sie mit der Seife aneinander reiben. Schäumen Sie die Handrücken, die Finger und die Fingernägel ein. Schrubben Sie Ihre Hände mindestens 20 Sekunden lang. Spülen Sie Ihre Hände gut unter fließendem, sauberem Wasser ab. Trocknen Sie Ihre Hände mit einem sauberen Handtuch ab oder lassen Sie sie an der Luft trocknen.

Zusammenfassungen von Text 5

Gold-Standard:

Händewaschen ist die effektivste Methode, um die Verbreitung von Keimen zu verhindern. Dazu sollte man die Hände mit kaltem Wasser waschen, dann Seife hinzufügen, 20 Sekunden lang ordentlich reiben und anschließend unter kaltem Wasser abspülen.

BERT (monolingual):

Saubere Hände können die Verbreitung von Keimen verhindern. Aber wie kann das sein? Ein paar Tipps, wie Sie die Verbreitung der Keimen vermeiden. Ein Überblick.

BERT (multilingual):

Keine Keime, keine Keime: Kein Wunder, dass Keime die Ausbreitung von Keimen zwischen Menschen und innerhalb einer ganzen Gemeinschaft verhindern. Wie man sich damit beschäftigen kann und wie man den Weg zu Keimen schützt.

XLM-R (monolingual):

Händewaschen ist eine der effektivsten Möglichkeiten, die Verbreitung von Keimen zu verhindern. Dabei ist es eine effektive Methode, um die Verbündeten zu bewahren und die Ausbrüche zu vermeiden. Ein paar einfache Schritte.

XLM-R (multilingual):

Handwaschen ist eine der effektivsten Möglichkeiten, die Verbreitung von Keimen zu verhindern. Ein Überblick über die effektivsten Mittel. Zumindest, wenn die Händewaschen zu sauber sind und die Hände mit sauberem Wasser befüllt sind. Die wichtigsten Tipps zum Umgang mit Keimen.

BART (multilingual):

Sauberes Händewaschen kann die Ausbreitung von Keimen zwischen Personen verhindern. Befolgen Sie stets diese fünf Schritte.

Quellcode

Konfigurationsdatei

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

ratio_corpus_wik: float = 1.0
ratio_corpus_nws: float = 1.0
ratio_corpus_mls: float = 1.0
ratio_corpus_eng: float = 1.0

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

train_size : float = 0.900
val_size : float = 0.025
test_size : float = 0.075

'''
- bert-base-multilingual-cased
- deepset/gbert-base
- xlm-roberta-base
- facebook/mbart-large-cc25
'''
```

Quelltext B.1: Konfigurationsdatei

Hilfsmethoden

```
# Imports
import csv
import datasets
import gc
import matplotlib.pyplot as plt
import psutil
import pandas as pd
import string
import torch
import transformers
import tf2tf_gpu_config as config

from collections import Counter
from datasets import ClassLabel
from nltk import ngrams
from typing import List, Tuple


# Methods
def load_data() -> Tuple[datasets.Dataset, datasets.Dataset, datasets.Dataset]:
    if config.language == "english":
        return load_english_data()

    if config.language == "german":
        return load_german_data()

    if config.language == "multilingual":
        return load_multilingual_data()
```

```
35 def load_english_data()
36   -> Tuple[datasets.Dataset, datasets.Dataset, datasets.Dataset]:
37     train_data = datasets.load_dataset(
38         "cnn_dailymail", "3.0.0", split="train",
39          ignore_verifications=True
40     )
41
42     val_data = datasets.load_dataset(
43         "cnn_dailymail", "3.0.0", split="validation[:50%]",
44          ignore_verifications=True
45     )
46
47     test_data = datasets.load_dataset(
48         "cnn_dailymail", "3.0.0", split="test[:50%]",
49          ignore_verifications=True
50     )
51
52     train_data = train_data.select(
53         range(0, int(len(train_data) * config.ratio_corpus_eng))
54     )
55
56     train_data = train_data.rename_column("article", "text")
57     train_data = train_data.rename_column("highlights", "summary")
58     train_data = train_data.remove_columns("id")
59
60     val_data = val_data.rename_column("article", "text")
61     val_data = val_data.rename_column("highlights", "summary")
62     val_data = val_data.remove_columns("id")
63
64     test_data = test_data.rename_column("article", "text")
65     test_data = test_data.rename_column("highlights", "summary")
66     test_data = test_data.remove_columns("id")
67
68     return train_data.shuffle(), val_data.shuffle(), test_data.shuffle()
69
70
```

```
71 def load_german_data()
72   -> Tuple[datasets.Dataset, datasets.Dataset, datasets.Dataset]:
73     ds_wik = load_corpus_wik()
74     ds_nws = load_corpus_nws()
75     ds_mls = load_corpus_mls()
76
77     german_data = datasets.concatenate_datasets([
78         ds_wik.select(
79             range(0, int(len(ds_wik) * config.ratio_corpus_wik))),
80         ds_nws.select(
81             range(0, int(len(ds_nws) * config.ratio_corpus_nws))),
82         ds_mls.select(
83             range(0, int(len(ds_mls) * config.ratio_corpus_mls)))
84     ])
85
86     train_size = int(len(german_data) * config.train_size)
87     valid_size = int(len(german_data) * config.val_size)
88     test_size = int(len(german_data) * config.test_size)
89
90     train_data = german_data.select(
91         range(0, train_size)
92     )
93
94     val_data = german_data.select(
95         range(train_size, train_size + valid_size)
96     )
97
98     test_data = german_data.select(
99         range(train_size + valid_size, train_size + valid_size + test_size)
100    )
101
102    return train_data, val_data, test_data
103
104
105
106
```

```
107 def load_corpus_wik() -> datasets.Dataset:
108     data_txt, data_ref = [], []
109 
110     with open("./data_train.csv", "r", encoding="utf-8") as f:
111         reader = csv.reader(f, delimiter=",", quoting=csv.QUOTE_ALL)
112         next(reader, None)
113 
114         for row in reader:
115             data_txt.append(row[0])
116             data_ref.append(row[1])
117 
118     df_wik = pd.DataFrame(
119         list(zip(data_txt, data_ref)),
120         columns=["text", "summary"]
121     )
122 
123     ds_wik = datasets.arrow_dataset.Dataset.from_pandas(df_wik)
124 
125     return ds_wik.shuffle()
126 
127 
128 def load_corpus_nws() -> datasets.Dataset:
129     df_nws = pd.read_excel("./data_train_test.xlsx", engine="openpyxl")
130     df_nws = df_nws[["article", "highlights"]]
131     df_nws.columns = ["text", "summary"]
132     df_nws = df_nws[~df_nws["summary"].str.contains("ZEIT")]
133     df_nws = df_nws.dropna()
134     ds_nws = datasets.arrow_dataset.Dataset.from_pandas(df_nws)
135     ds_nws = ds_nws.remove_columns("__index_level_0__")
136 
137     return ds_nws.shuffle()
138 
139 
140 
141 
142 
```

```
143 def load_corpus_mls() -> datasets.Dataset:
144     ds_mls = datasets.load_dataset("mlsum", "de", split="train")
145     ds_mls = ds_mls.remove_columns(["topic", "url", " title ", "date"])
146 
147     text_corpus_mls = []
148     summary_corpus_mls = []
149 
150     for entry in ds_mls:
151         text = entry["text"]
152         summary = entry["summary"]
153 
154         if summary in text:
155             text = text[len(summary) + 1:len(text)]
156 
157         text_corpus_mls.append(text)
158         summary_corpus_mls.append(summary)
159 
160     df_mls = pd.DataFrame(
161         list (zip(text_corpus_mls, summary_corpus_mls)),
162         columns=["text", "summary"]
163     )
164 
165     ds_mls = datasets.arrow_dataset.Dataset.from_pandas(df_mls)
166 
167     return ds_mls. shuffle ()
168 
169 
170 
171 
172 
173 
174 
175 
176 
177 
178 
```

```
179 def load_multilingual_data()
180   -> Tuple[datasets.Dataset, datasets.Dataset, datasets.Dataset]:
181     english_data, _, _ = load_english_data()
182     german_data, _, _ = load_german_data()
183 
184     multilingual_data = datasets.concatenate_datasets([
185         german_data, english_data
186     ]).shuffle()
187 
188     train_size = int(len(multilingual_data) * config.train_size)
189     valid_size = int(len(multilingual_data) * config.val_size)
190     test_size = int(len(multilingual_data) * config.test_size)
191 
192     train_data = multilingual_data.select(
193         range(0, train_size)
194     )
195 
196     val_data = multilingual_data.select(
197         range(train_size, train_size + valid_size)
198     )
199 
200     test_data = multilingual_data.select(
201         range(train_size + valid_size, train_size + valid_size + test_size)
202     )
203 
204     return train_data, val_data, test_data
205 
206 
207 def test_cuda() -> None:
208     device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
209     torch.cuda.empty_cache()
210 
211     print("Device:", device)
212     print("Version:", torch.__version__)
213 
214 
```

```
215 def explore_corpus(data: datasets.Dataset) -> None:
216     df = pd.DataFrame(data)
217 
218     text_list = []
219     summary_list = []
220 
221     for index, row in df.iterrows():
222         text = row["text"]
223         summary = row["summary"]
224         text_list.append(len(text))
225         summary_list.append(len(summary))
226 
227     df = pd.DataFrame(data[:1])
228 
229     for column, typ in data.features.items():
230         if isinstance(typ, ClassLabel):
231             df[column] = df[column].transform(lambda i: typ.names[i])
232 
233 
234 def empty_cache() -> None:
235     gc.collect()
236     torch.cuda.empty_cache()
237     psutil.virtual_memory()
238 
239 
240 
241 
242 
243 
244 
245 
246 
247 
248 
249 
250 
```

```
251 def load_tokenizer_and_model(from_checkpoint: bool = False)
252   -> Tuple[transformers.AutoTokenizer, transformers.EncoderDecoderModel]:
253     tokenizer = transformers.AutoTokenizer.from_pretrained(
254         config.tokenizer_name, strip_accent=False
255     )
256
257     if from_checkpoint:
258         if "mbart" in config.model_name:
259             tf2tf = transformers.AutoModelForSeq2SeqLM.from_pretrained(
260                 config.path_checkpoint
261             )
262
263         else:
264             tf2tf = transformers.EncoderDecoderModel.from_pretrained(
265                 config.path_checkpoint
266             )
267
268     else:
269         if "mbart" in config.model_name:
270             tf2tf = transformers.MBartForConditionalGeneration.from_pretrained(
271                 config.model_name
272             )
273
274         else:
275             tf2tf = transformers.EncoderDecoderModel.
                 from_encoder_decoder_pretrained(
276                 config.model_name, config.model_name, tie_encoder_decoder=True
277             )
278
279     return tokenizer, tf2tf
280
281
282
283
284
285
```

```
286 def configure_model(tf2tf: transformers.EncoderDecoderModel, tokenizer:
        transformers.AutoTokenizer)
287   -> transformers.EncoderDecoderModel:
288     tf2tf.config.decoder_start_token_id = tokenizer.cls_token_id
289     tf2tf.config.bos_token_id = tokenizer.bos_token_id
290     tf2tf.config.eos_token_id = tokenizer.sep_token_id
291     tf2tf.config.pad_token_id = tokenizer.pad_token_id
292
293     tf2tf.config.max_length = 128
294     tf2tf.config.min_length = 56
295     tf2tf.config.no_repeat_ngram_size = 3
296     tf2tf.config.early_stopping = True
297     tf2tf.config.length_penalty = 2.0
298     tf2tf.config.num_beams = 2
299
300     return tf2tf
```

Quelltext B.2: Hilfsmethoden

Trainingscode

```
# Imports
import datasets
import transformers
import tf2tf_gpu_config as config
import tf2tf_gpu_helpers as helpers


# Main
tokenizer, tf2tf = helpers.load_tokenizer_and_model(from_checkpoint=False)
train_data, val_data, test_data = helpers.load_data()
rouge = datasets.load_metric("rouge")

helpers.test_cuda()
helpers.explore_corpus(train_data)
helpers.empty_cache()

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()
    batch["labels"] = [[-100 if token == tokenizer.pad_token_id else token
                        for token in labels] for labels in batch["labels"]]

    return batch


train_data = train_data.map(
    process_data_to_model_inputs,
    batched=True,
    batch_size=config.batch_size,
    remove_columns=["text", "summary"]
)

train_data.set_format(
    type="torch",
    columns=["input_ids",
             "attention_mask",
             "decoder_input_ids",
             "decoder_attention_mask",
             "labels"]
)
```

```
val_data = val_data.map(
    process_data_to_model_inputs,
    batched=True,
    batch_size=config.batch_size,
    remove_columns=["text", "summary"]
)

val_data.set_format(
    type="torch",
    columns=["input_ids",
             "attention_mask",
             "decoder_input_ids",
             "decoder_attention_mask",
             "labels"]
)


def compute_metrics(pred):
    labels_ids = pred.label_ids
    pred_ids = pred.predictions

    pred_str = tokenizer.batch_decode(pred_ids, skip_special_tokens=True)
    labels_ids[labels_ids == -100] = tokenizer.pad_token_id
    label_str = tokenizer.batch_decode(labels_ids, skip_special_tokens=True)

    rouge_output = rouge.compute(
        predictions=pred_str,
        references=label_str,
        rouge_types=["rouge2"]
    )["rouge2"].mid

    return {
        "rouge2_precision": round(rouge_output.precision, 4),
        "rouge2_recall": round(rouge_output.recall, 4),
        "rouge2_fmeasure": round(rouge_output.fmeasure, 4),
    }
```

```
107 if "mbart" in config.model_name:
108     training_args = transformers.TrainingArguments(
109         output_dir=config.path_output,
110         logging_dir=config.path_output,
111         per_device_train_batch_size=1,
112         per_device_eval_batch_size=1,
113         num_train_epochs=1,
114         warmup_steps=500,
115         weight_decay=0.01
116     )
117
118     trainer = transformers.Trainer(
119         model=tf2tf,
120         args=training_args,
121         train_dataset=train_data,
122         eval_dataset=val_data
123     )
124
125 else:
126     training_args = transformers.Seq2SeqTrainingArguments(
127         predict_with_generate=True,
128         evaluation_strategy="steps",
129         per_device_train_batch_size=config.batch_size,
130         per_device_eval_batch_size=config.batch_size,
131         output_dir=config.path_output,
132         warmup_steps=1000,
133         save_steps=10000,
134         logging_steps=2000,
135         eval_steps=10000,
136         save_total_limit=1,
137         learning_rate=5e-5,
138         adafactor=True,
139         fp16=True
140     )
141
142
```

```
143     trainer  = transformers.Seq2SeqTrainer(
144         model=tf2tf,
145         args=training_args,
146         compute_metrics=compute_metrics,
147         train_dataset=train_data,
148         eval_dataset=val_data,
149         tokenizer=tokenizer
150     )
151
152 trainer.train()
```

Quelltext B.3: Trainingscode

Evaluationscode

```
# Imports
import datasets
import transformers
import tf2tf_gpu_config as config
import tf2tf_gpu_helpers as helpers


# Main
rouge = datasets.load_metric("rouge")

if "mbart" in config.model_name:
    data, _, _ = helpers.load_data()
    model = transformers.pipeline("summarization")

     list_of_texts = list(data["text"])
     list_of_candidates = list(data["summary"])
    list_of_summaries, list_of_predictions = [], []

    for i in range(0, len( list_of_texts ) - 1):
        try:
            summary = model(
                str( list_of_texts [i]) [0:1024],
                max_length=156,
                min_length=64,
                do_sample=False
            ) [0]["summary_text"]

             list_of_predictions .append(summary)
            list_of_summaries.append(list_of_candidates[i])

        except Exception as e:
            print(e)


```

```
    print(
        rouge.compute(
            predictions= list_of_predictions ,
            references=list_of_summaries,
            rouge_types=["rouge2"]
        )["rouge2"].mid
    )

else:
    tokenizer, tf2tf = helpers.load_tokenizer_and_model(from_checkpoint=True)
    train_data, val_data, test_data = helpers.load_data()

    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
```

```
71      results  = test_data.map(
72          generate_summary,
73          batched=True,
74          batch_size=config.batch_size
75      )
76
77      print(
78          rouge.compute(
79              predictions=results["pred_summary"],
80              references=results["summary"],
81              rouge_types=["rouge2"]
82          )["rouge2"].mid
83      )
```

Quelltext B.4: Evaluationscode

Beispielcode

```
# Installation
%%capture

!pip install transformers==4.5.1
!pip install datasets==1.6.2
!pip install tokenizers==0.10.2
!pip install torch==1.8.1+cu111
!pip install psutil==5.8.0
!pip install rouge_score
!pip install sacrebleu
!pip install openpyxl
!pip install xlrd
!pip install git-python
!pip install -U ipython==7.20
!pip install cmake
!pip install SentencePiece


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

from datasets import ClassLabel
from IPython.display import display, HTML
from typing import Tuple



```

```
# Drive
from google.colab import drive
drive.mount("/content/drive")
path_drive = "/content/drive/My Drive/Temp/"


# Methods
def split_long_texts(parts: List[str], text: str):
    limit = 512

    if len(text) > limit:
        end_index = max([
            text.rfind(".", 0, limit),
            text.rfind("!", 0, limit),
            text.rfind("?", 0, limit)
        ])

        parts.append(text[0:end_index + 1].strip())
        text = text[end_index + 1:len(text)].strip()
        parts = split_long_texts(parts, text)

    else:
        parts.append(text)

    return parts
```

```
71 # Example (BERT, XLM-R)
72 tokenizer, tf2tf = load_tokenizer_and_model(from_checkpoint=True)
73
74 if language == "german":
75     corpus = corpus_german
76
77 if language == "english":
78     corpus = corpus_english
79
80 if language == "multilingual":
81     corpus = corpus_english + corpus_german
82
83 tf2tf = configure_model(tf2tf, tokenizer)
84 tf2tf.to("cuda")
85
86 test_cuda()
87 empty_cache()
88
89 cnt = 0
90
91 for text in corpus:
92     cnt += 1
93     parts = split_long_texts([], text)
94
95     if len(parts) > 1:
96         article = parts
97         highlights = [None] * len(parts)
98
99     else:
100        parts = [text]
101        article = [text] * 2
102        highlights = [None] * 2
103
104    df = pd.DataFrame({"text": article, "summary": highlights})
105    test_data = datasets.arrow_dataset.Dataset.from_pandas(df)
106
```

```
107     def generate_summary(batch):
108         inputs = tokenizer(
109             batch["text"],
110             padding="max_length",
111             truncation=True,
112             max_length=512,
113             return_tensors="pt"
114         )
115 
116         input_ids = inputs.input_ids.to("cuda")
117         attention_mask = inputs.attention_mask.to("cuda")
118 
119         outputs = tf2tf.generate(input_ids, attention_mask=attention_mask)
120         output_str = tokenizer.batch_decode(outputs, skip_special_tokens=True)
121         batch["pred_summary"] = output_str
122 
123         return batch
124 
125 
126     summary = test_data.map(
127         generate_summary,
128         batched=True,
129         batch_size=batch_size
130     )
131 
132     result = ""
133 
134     for i in range(0, len(parts)):
135         result = result + " " + summary[i]["pred_summary"]
136 
137     with open(path_output + "summary_" + language + "_" + model_name + "_text
        –" + str(cnt) + ".txt", "w", encoding="utf–8") as f:
138         f.write(summary[0]["pred_summary"])
139 
140 
141 
```

```
142 # Example (BART)
143 model = transformers.pipeline("summarization")
144 corpus = corpus_english + corpus_german
145
146 cnt = 0
147
148 for text in corpus:
149     cnt += 1
150
151     summary = model(
152         text,
153         max_length=156,
154         min_length=64,
155         do_sample=False
156     ) [0]["summary_text"]
157
158     with open(path_output + "summary_" + language + "_BART_text-" + str(cnt)
        + ".txt", "w", encoding="utf-8") as f:
159         f.write(summary)
```

Quelltext B.5: Beispielcode

Datenexploration

```
# Imports
import datasets
import transformers
import tf2tf_gpu_config as config
import tf2tf_gpu_helpers as helpers


# Methods
def clean(text: str) -> str:
    text = " ".join([w.lower() for w in text.split()])
    text = text.translate(str.maketrans("", "", string.punctuation))

    return text


def analyze_text_lenghts(corpus: List[str]) -> None:
    lengths = []

    for text in corpus:
        lengths.append(len(text.split()))

    fig = plt.figure(figsize=(10, 6))

    plt.hist(lengths, bins=40)
    plt.xlabel("Anzahl der Woerter", fontsize=18)
    plt.ylabel("Anzahl der Texte", fontsize=18)
    plt.xticks(rotation=0, fontsize=12)
    plt.yticks(rotation=0, fontsize=12)
    plt.tight_layout()

    plt.show()


```

```
35 def analyze_most_common_words(corpus: List[str], k: int = 20) -> None:
36     words = []
37 
38     for text in corpus:
39         token_list = clean(text).split()
40 
41         for token in token_list:
42             if len(token) > 1:
43                 words.append(token)
44 
45     most_common_words = Counter(words).most_common(k)
46 
47     x = [tuple[0] for tuple in most_common_words]
48     y = [tuple[1] for tuple in most_common_words]
49 
50     fig = plt.figure(figsize=(10, 6))
51 
52     plt.bar(x, y)
53     plt.xlabel("Woerter", fontsize=18)
54     plt.ylabel("Anzahl", fontsize=18)
55     plt.xticks(rotation=90, fontsize=12)
56     plt.yticks(rotation=0, fontsize=12)
57     plt.tight_layout()
58 
59     plt.show()
60 
61 
62 def analyze_n_grams(corpus: List[str], n: int = 3, n_most_common_n_grams: int =
       20) -> None:
63     n_grams = []
64 
65     for text in corpus:
66         try:
67             for n_gram in ngrams(text.split(), n):
68                 n_grams.append(n_gram)
69 
```

```
70            except Exception as e:
71                print(e)
72
73        most_common_n_grams = Counter(n_grams).most_common(
              n_most_common_n_grams)
74
75        x = [" ".join(tuple[0]) for tuple in most_common_n_grams]
76        y = [tuple[1] for tuple in most_common_n_grams]
77
78        fig = plt.figure( figsize =(10, 6))
79
80        plt.bar(x, y)
81        plt.xlabel("N-Gramme", fontsize=18)
82        plt.ylabel("Anzahl", fontsize=18)
83        plt.xticks(rotation=90, fontsize=12)
84        plt.yticks(rotation=0, fontsize=12)
85        plt.tight_layout()
86
87        plt.show()
88
89
90    # Main
91    data, _, _ = load_data()
92    corpus = list(data["text"])
93
94    analyze_text_lenghts(corpus)
95    analyze_most_common_words(corpus)
96    analyze_n_grams(corpus, n=2)
97    analyze_n_grams(corpus, n=3)
```

Quelltext B.6: Datenexploration

Literaturverzeichnis

[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 19.05.2021.

[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 16.04.2021.

[Culurciello, 2018] Culurciello, Eugenio: The Fall of RNN/ LSTM, in: https://towardsdatascience.com/the-fall-of-rnn-lstm-2d1594c74ce0, Aufruf am 23.04.2021.

[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 26.05.2021.

[Edpresso, O. J.] Edpresso: Overfitting and Underfitting, in: https://www.educative.io/edpresso/overfitting-and-underfitting, Aufruf am 09.04.2021.

[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 16.03.2021.

[Goodfellow et al., 2016] Goodfellow, Ian & Bengio, Yoshua & Courville, Aaron: Deep Learning, in: https://www.deeplearningbook.org/, Aufruf am 03.07.2021.

[HuggingFace, 2021] HuggingFace: The AI community building the future, in: https://huggingface.co/, Aufruf am 14.06.2021.

[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 19.05.2021.

[Irene, 2018] Irene: ELMo in Practice, in: https://ireneli.eu/2018/12/17/elmo-in-practice/, Aufruf am 26.05.2021.

[Karani, 2018] Karani, Dhruvil: Introduction to Word Embedding and Word2Vec, in: https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa, Aufruf am 19.05.2021.

[Karim, 2019] Karim, Raimi: Self-Attention, in: https://towardsdatascience.com/illustrated-self-attention-2d627e33b20a, Aufruf am 26.05.2021.

[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.

[Lemberger, 2020] Lemberger, Pirmin: Deep Learning Models for Automatic Summarization, in: https://towardsdatascience.com/deep-learning-models-for-automatic-summarization-4c2b89f2a9ea, Aufruf am 26.05.2021.

[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.

[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 09.04.2021.

[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 19.05.2021.

[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 19.05.2021.

[NVIDIA, 2021] NVIDIA: CUDA Toolkit, in: https://developer.nvidia.com/cuda-toolkit, Aufruf am 14.06.2021.

[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 16.03.2021.

[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 19.05.2021.

[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 19.05.2021.

[Sprachenfabrik, 2016] Sprachenfabrik: Eigenarten der deutschen Sprache, in: https://sprachenfabrik.de/de/2016/01/18/7-eigenarten-der-deutschen-sprache/, Aufruf am 07.07.2021.

[TensorFlow, 2021] TensorFlow: Datasets - CNN-DailyMail, in: https://www.tensorflow.org/datasets/catalog/cnn_dailymail, Aufruf am 23.03.2021.

[Thaker, 2019] Thaker, Madhav: Comparing Text Summarization Techniques, in: https://towardsdatascience.com/comparing-text-summarization-techniques-d1e2e465584e, Aufruf am 23.03.2021.

[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 16.03.2021.

[Wissler et al., 2014] Wissler, Lars & Almashraee, Mohammed & Monett, Dagmar & Pasche, Adrian: The Gold Standard in Corpus Annotation, Free University, Berlin, 2014.

[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 09.04.2021.

[ZIH, 2021] Zentrum für Informationsdienste und Hochleitungsrechnen der TU Dresden: Hochleitungsrechnen, in: https://tu-dresden.de/zih/hochleistungsrechnen, Aufruf am 14.06.2021.

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 8. August 2021

Daniel Vogel