Hochschule für Technik und Wirtschaft Dresden
Fakultät Informatik/ Mathematik

Abschlussarbeit zur Erlangung des akademischen Grades

Master of Science

Thema:

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

eingereicht von: Daniel Vogel

eingereicht am: 13. August 2021

Erstgutachter: Prof. Dr. Hans-Joachim Böhme

Zweitgutachter: Dipl.-Kfm. Torsten Rex

Autorenreferat

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

Abstract

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

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ADAM	Adaptive Momentum Estimation
ATS	Automatic Text Summarization
BERT	Bidirectional Encoder Representations from Transformers
BOW	Bag-of-Words
DL	Deep Learning
ELMo	Embeddings from Language Models
GRU	Gated Recurrent Units
LR	Learning Rate
LSTM	Long-Short-Term-Memory-Networks
ML	Machine Learning
MLP	Multi-Layer-Perceptron
NLG	Natural Language Generation
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
NLU	Natural Language Understanding
OHE	One-Hot-Encoding
RL	Reinforcement Learning
RNN	Recurrent Neural Networks
SOTA	State-of-the-Art
TL	Transfer Learning
W2V	Word2Vec

Quellcodeverzeichnis

1 Einleitung

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

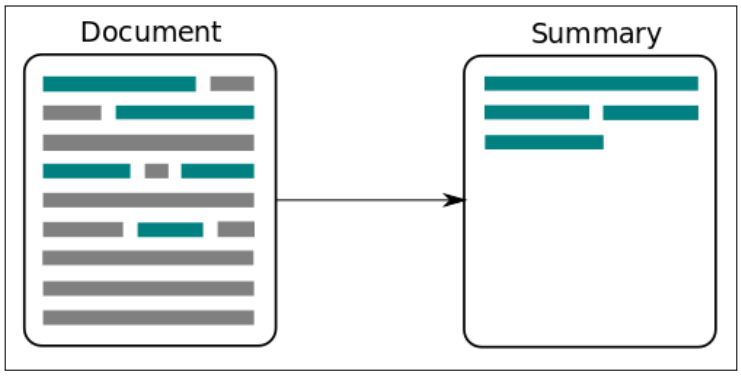

Abbildung 1.1: Ablauf einer automatischen Zusammenfassung [Thaker, 2019].

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

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

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

1.1 Zielsetzung

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

Die Forschungsfragen lauten wie folgt:

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

1.2 Aufbau der Arbeit

Nach der Einleitung werden zunächst die Grundlagen des Deep Learning (DL) und des NLP offengelegt. Im Kapitel des DL werden neuronale Netze als solches definiert und ausgewählte Architekturen, welche auf die Zielerreichung einwirken, vorgestellt. Die Eigenschaften und die Relevanz von Hyperparametern und von TL schließen sich an. Im Kapitel des NLP werden neben der prinzipiellen Arbeit mit natürlicher Sprache und der entsprechenden Vorverarbeitung insbesondere sogenannte Word Embeddings und Deep Language Representations thematisiert.

Bevor die bis dahin behandelten Komponenten in ein tatsächliches Modell integriert werden können, ist die Beschreibung der Datengrundlage erforderlich. Zum daran anschließenden abstraktiven Ansatz gehört die Erläuterung der Architektur, die Beschreibung des Trainingsprozesses und die Evaluation der Ergebnisse. Bei der sprachtechnischen Adaption des Modells auf die deutsche Sprache werden zuerst entsprechende Anpassungen an der ursprünglichen Architektur konzipiert, bevor erneut der Trainingsprozess beschrieben wird und die dazugehörigen Ergebnisse evaluiert werden. Der entsprechende Quellcode wird in Python entwickelt.

1.3 Forschungsstand & Referenzen

Aufgrund der stetig fortschreitenden Entwicklungen überholt sich der Forschungsstand der ATS regelmäßig. Dennoch haben sich in den vergangenen Jahren gewisse Tendenzen erkennen lassen. Bereits zur Jahrtausendwende existierten erste ATS-Systeme. Waren die ersten Ansätze zumeist noch extraktiv, wurde sich in den vergangenen Jahren mehr und mehr auf die abstraktiven Ansätze konzentriert. Vor 2016 schienen Ansätze mit Recurrent Neural Networks (RNN) und Long-Short-Term-Memory-Networks (LSTM) sehr populär [Nallapati et al., 2016]. In den Jahren 2016 und 2017 etablierten sich Ansätze, welche auf Reinforcement Learning (RL) basierten [Paulus et al., 2017]. Seit 2018 legten diverse Ansätze mit Encoder-Decoder-Architekturen die Grundlage des heutigen

State-of-the-Art (SOTA) [Yang et al., 2019, Rothe et al., 2020], denn um den SOTA konkurrieren fast ausschließlich sogenannte Transformer. Diese basieren auf den Encoder-Decoder-Architekturen, implementieren verschiedenartige Attention-Mechanismen und haben sich sowohl unter qualitativen als auch unter ökonomischen und ökologischen Aspekten bewiesen [Zhang et al., 2020]. Diese Arbeit wird daher ebenfalls diesen Ansatz verfolgen.

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

Weiterhin hat der Durchbruch frei verfügbarer vortrainierter Modelle die NLP-Welt revolutioniert, wie beispielsweise Bidirectional Encoder Representations from Transformers (BERT) [Devlin et al., 2019] oder auch Embeddings from Language Models (ELMo) [Peters et al., 2018] sowie deren Weiterentwicklungen. Verschiedenste NLP-Aufgaben wie die ATS konnten hiervon sehr stark profitieren. Die konkreten Funktionsweisen werden ebenfalls im Verlauf dieser Arbeit offengelegt. Wissenschaftliche Publikationen, welche mit dieser Arbeit vergleichbar sind und in dieser Arbeit oftmals referenziert werden, lauten wie folgt:

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

2 Deep Learning

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

2.1 Neuronale Netze

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

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

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

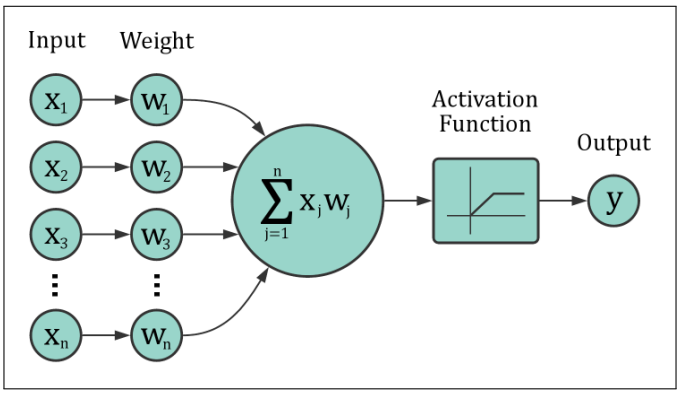

Abbildung 2.1: Aufbau eines künstlichen Neurons [McCullum, 2020].

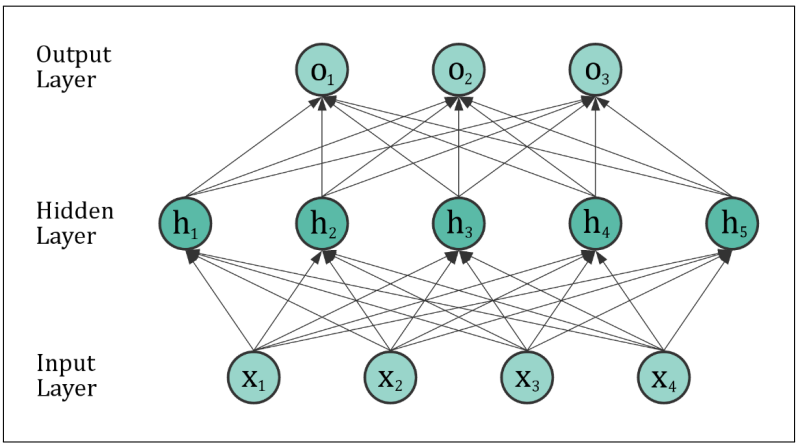

Abbildung 2.2: Aufbau eines MLP [Zhang et al., 2020, S. 133].

Ziel der hoch parametrisierten neuronalen Netze ist es, komplexe Funktionen hohen Grades bestmöglich zu approximieren und so verschiedenste Probleme zu lösen. Der anvisierte Lerneffekt wird mithilfe des sogenannten Backpropagation-Algorithmus erreicht. Hierbei werden Eingangsdaten zunächst vorwärts durch ein neuronales Netz hindurch propagiert. Mithilfe einer Fehlerfunktion wird sodann die erwartete mit der tatsächlichen Ausgabe verglichen und bewertet. 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 [Zhang et al., 2020, S. 140, 169].

Der Trainingsprozess erfolgt optimalerweise über mehrere sogenannte Epochen. Hier werden dem neuronalen Netz verschiedene Eingangsdaten zugeführt und beidseitige Propagationen ausgeführt. Wichtig ist dennoch, kein Overfitting beziehungsweise Underfitting zu erzeugen. Dies würde bedeuten, dass das trainierte Modell zu sehr oder zu wenig auf die Trainingsdaten angepasst ist. Ziel ist ein möglichst hoher Generalisierungseffekt des Modells, wie Abbildung 2.3 zeigt. Das Modell sollte den Lernfortschritt auf unbekannte Daten adaptieren können und darauf eine hohe Genauigkeit erreichen. Es gibt verschiedene Ansätze, um beispielsweise Overfitting vorzubeugen. Hier seien insbesondere Batch Normalization, Dropout und Early Stopping genannt, wobei entsprechende Mechanismen an anderweitiger Stelle erläutert werden [Zhang et al., 2020, S. 143-149].

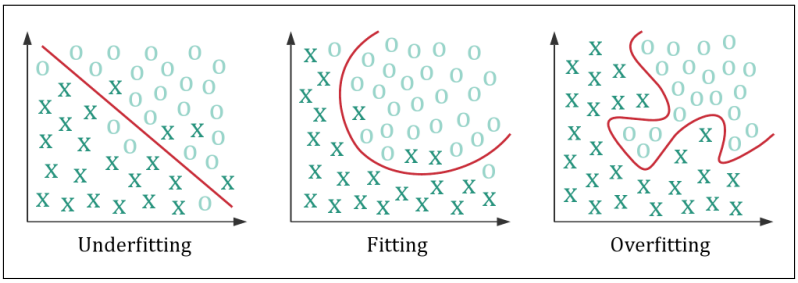

Abbildung 2.3: Typen von Generalisierungseffekten [Edpresso, O. J.].

2.2 Architekturen

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

2.2.1 Recurrent Neural Networks

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

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

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

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

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

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

Abbildung 2.4 demonstriert den rechentechnischen Vorgang anhand von drei prototypischen verborgenen Zuständen unter gegebenen Eingangsdaten. Wie bereits bei der oben genannten Funktion ersichtlich wurde, gehen in einen verborgenen Zustand jeweils das momentane Wort und der vorhergegangene verborgene Zustand ein. Innerhalb eines verborgenen Zustandes werden diese beiden Eingangsgrößen mithilfe einer Aktivierungsfunktion in eine vollvermaschte Schicht überführt. Zudem wird die Ausgabeschicht mit dem momentanen Output versorgt [Zhang et al., 2020, S. 325]. Allgemein haben RNN also die Form einer Kette, in der Module neuronaler Netze immer wiederkehren [AI-United, 2019].

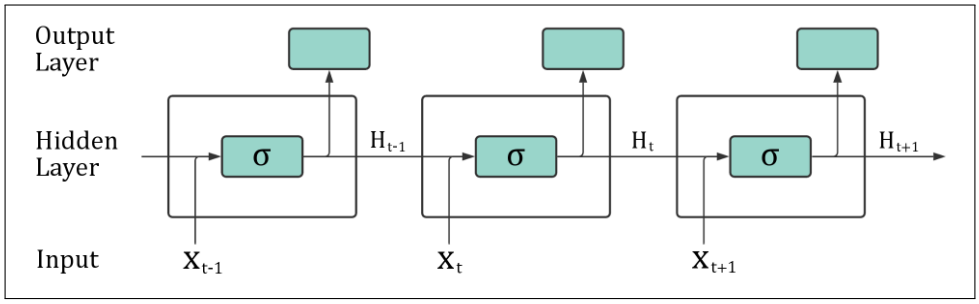

Abbildung 2.4: RNN mit verborgenen Zuständen [Zhang et al., 2020, S. 325].

Die bereits bekannte beidseitige Propagation ist auch bei RNN erforderlich, damit der Trainingsprozess zu einem entsprechenden Lernfortschritt führt. Während die vorwärtige Propagation den oben beschriebenen Prinzipien nachkommt, wird die Backpropagation nun über die Indizes hinweg durchgeführt (engl. Backpropagation Through Time). Bei

einer langen Eingabesequenz gehen entsprechend hochdimensionale Matrizen in die Berechnung ein. Dies ist sowohl aus rechentechnischer als auch aus numerischer Sicht nachteilig. Die Anzahl modellinterner Parameter steigt bei steigender Indexanzahl hingegen nicht [Zhang et al., 2020, S. 328, 340].

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

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

LSTM sind eine Art von RNN, welche dazu fähig sind, langfristige Abhängigkeit zu erlernen. Sie vermeiden das Problem verschwindender Gradienten [AI-United, 2019]. Außerdem verfolgen sie das Prinzip sogenannter Gated Recurrent Units (GRU). Der entscheidende Mechanismus lässt aus der Funktionsweise von Gates ableiten. Demnach entscheiden diese Gates mit einer Wahrscheinlichkeit zwischen 0 und 1, ob der erhaltene Wert weitergeleitet oder verworfen werden soll. Eine LSTM-Zelle verfügt über drei wesentliche Gates: Input Gate, Forget Gate, Output Gate [Zhang et al., 2020, S. 347-348].

Das Input Gate definiert, welche Werte in der Zelle berücksichtigt werden sollen. Hierzu gehören gemäß RNN das momentane Wort sowie der vorhergegangene verborgene Zustand. Das Forget Gate trifft eine Entscheidung darüber, ob und welche Werte gespeichert oder vergessen werden sollen. Hierfür eignen sich beispielsweise Aktivierungsfunktionen [Zhang et al., 2020, S. 355]. Über die intern festgelegte Logik kann somit der Zellzustand berechnet werden, unter anderem mithilfe punktweiser Matrixoperationen. Der Zellzustand wird sodann als neuer verborgener Zustand im Output Gate emittiert [Luber et al., 2018]. Der Aufbau einer LSTM-Zelle ist in Abbildung 2.5 zu sehen.

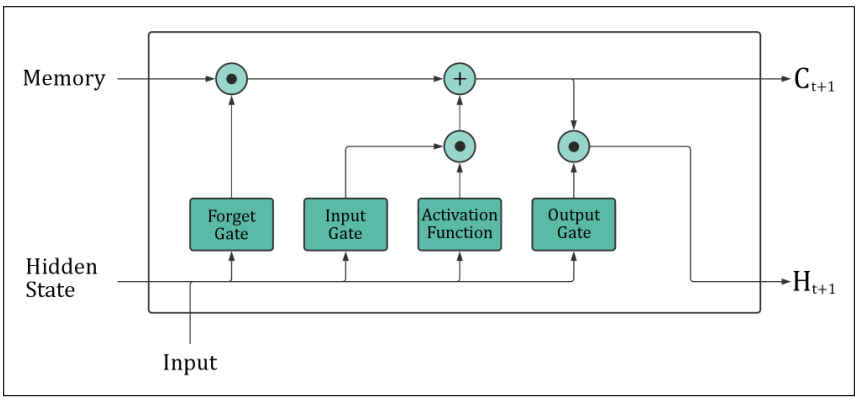

Abbildung 2.5: Aufbau einer LSTM-Zelle [Zhang et al., 2020, S. 357].

Ersetzt man die verborgenen Zustände in Abbildung 2.4 durch LSTM-Zellen, so wird man der beabsichtigten Funktionsweise eines LSTM gerecht. Mit fortschreitendem Trainingsprozess werden auch hier sequenzielle Abhängigkeiten gelernt, wobei als irrelevant angenommene Informationen stets verworfen werden. Der Vorteil gegenüber schlichter RNN besteht bei den LSTM hierdurch insbesondere darin, dass langfristige Abhängigkeiten besonders gut gelernt und bestimmte Informationen schneller gefunden werden können. Dies gilt vor allem für vielschichtige komplexe neuronale Netze, welche anderenfalls unübersichtlich und rechenaufwändig würden [Luber et al., 2018].

Zwar sind LSTM den RNN qualitativ in vielen NLP-Aufgaben überlegen, dennoch sind beide Architekturen eher statistisch getrieben und somit autark als veraltet zu bewerten. Obgleich sie als Grundlage prinzipiell verstanden werden sollten, dominieren maschinelle Lernverfahren qualitativ sowie ressourcentechnisch den SOTA mit verschiedenen anderweitigen Architekturen [Culurciello, 2018]. Diese werden nachfolgend hinreichend erläutert.

2.2.2 Encoder-Decoder-Networks

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

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

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

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

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

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

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

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

Um die theoretisch und abstrakt beschriebene Architektur zu veranschaulichen, werden die wesentlichen Module nun abschließend in Abbildung 2.6 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]. Zuletzt sei noch erwähnt, dass es Suchalgorithmen wie die Beam Search mithilfe geeigneter mathematischer Ansätze ermöglichen, Ausgabesequenzen mit variablen Längen zu erzeugen [Zhang et al., 2020, S. 387].

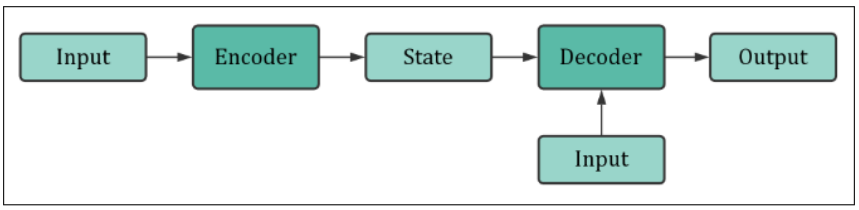

Abbildung 2.6: Encoder-Decoder-Architektur [Zhang et al., 2020, S. 375].

2.2.3 Attention in Neural Networks

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

Der Attention-Mechanismus basiert indes auf einer Attention-Funktion, welche eine Query und ein Menge von Key-Value-Paaren einem Output zuordnet. Der Output wird dabei vektorweise als gewichtete Summe der Werte berechnet, während die Query determiniert, welchen Werten mehr Attention zugeordnet wird [Vaswani et al., 2017]. Abbildung 2.7 zeigt eine solche Attention-Schicht.

Die Funktionsweise des Attention-Mechanismus innerhalb einer Encoder-Decoder-Architektur kann nun mithilfe Abbildung 2.8 weiter vertieft werden, wobei hier Index t betrachtet wird. Dabei bettet der Encoder die Eingabesequenz in bekannter Weise ein und verarbeitet sie, indem die verborgenen Zustände über alle verborgenen Schichten hinweg berechnet werden. Die Attention-Schicht erhält in der Folge alle Informationen, die der Encoder verarbeitet hat. Der Decoder wird nicht nur über den vorangegangenen verborgenen Zustand des Encoders informiert, sondern auch über den aus der Attention-Schicht

resultierenden Kontext. Dieser wird als Antwort auf eine Query generiert, wobei eben diese Query wiederum durch den vorangegangenen verborgenen Zustand des Decoders repräsentiert wird [Zhang et al., 2020, S. 394]. Die Ausgabesequenz wird hierbei indexweise und autoregressiv generiert, da dem Decoder in jedem Index zusätzlich die bereits generierten Wörter zugeführt werden [Vaswani et al., 2017].

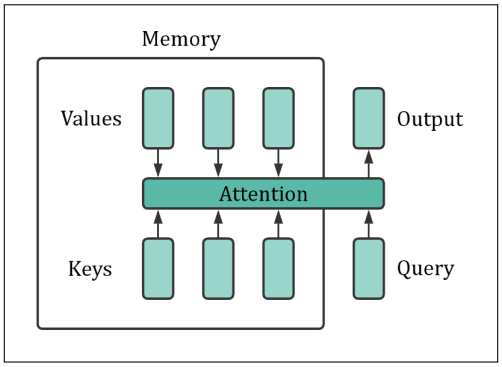

Abbildung 2.7: Attention-Schicht [Zhang et al., 2020, S. 390].

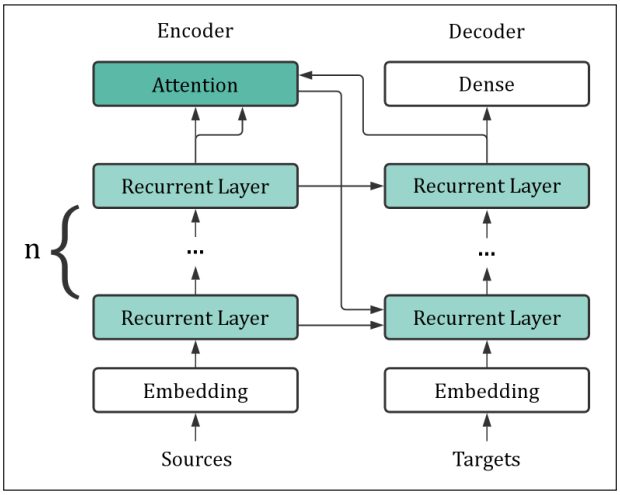

Abbildung 2.8: Attention-Mechanismus [Zhang et al., 2020, S. 394].

Weiterhin wird zwischen zwei Eigenarten unterschieden: Self-Attention und Multi-Head-Attention. Self-Attention transformiert innerhalb einer Query n Inputs in n Outputs. Dabei interagieren alle Inputs miteinander, um die Verteilung der globalen Attention zu bestimmen. Die Outputs entstehen folglich, indem die entsprechenden Scores aggregiert werden. Betrachtet man also einen Satz, dann ist die Attention jedes darin enthaltenen Wortes zu berechnen [Karim, 2019]. Abbildung 2.9 visualisiert die Self-Attention architektonisch.

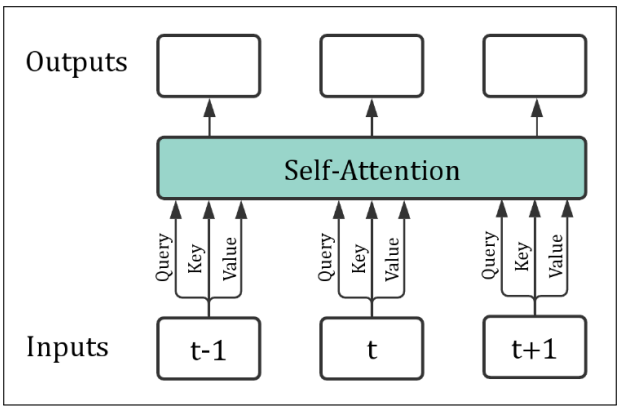

Abbildung 2.9: Self-Attention [Zhang et al., 2020, S. 400].

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

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

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

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

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

2.2.4 Transformer Networks

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

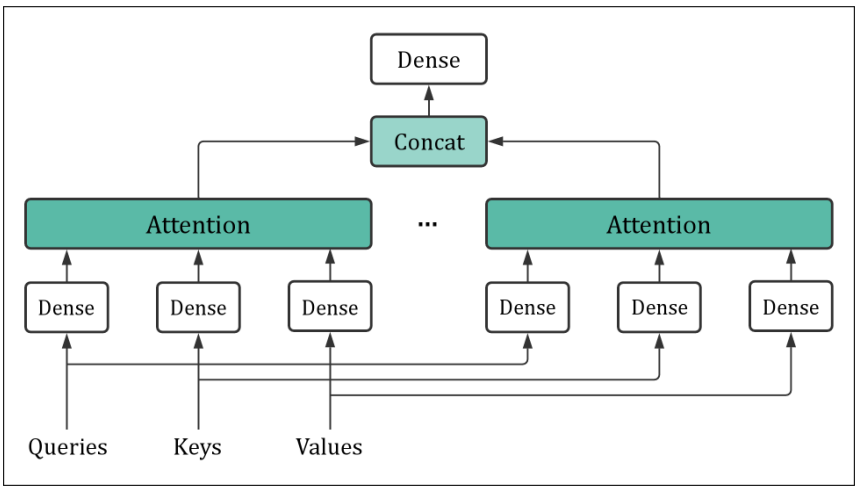

Abbildung 2.10: Multi-Head-Attention [Zhang et al., 2020, S. 400].

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

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.

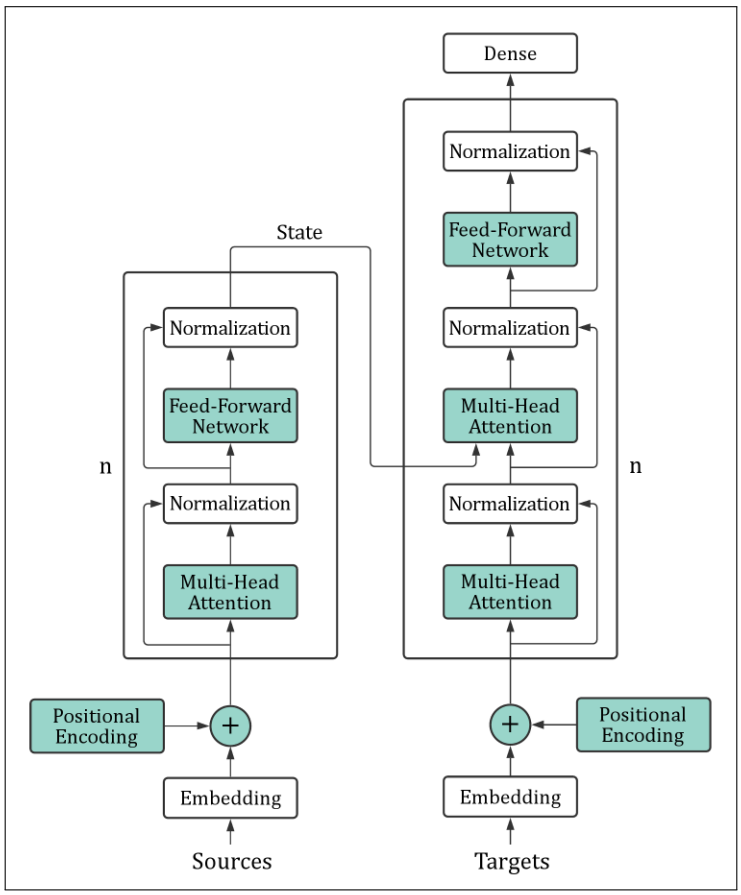

Abbildung 2.11: Transformer-Architektur [Zhang et al., 2020, S. 399].

2.3 Hyperparameter

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

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

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

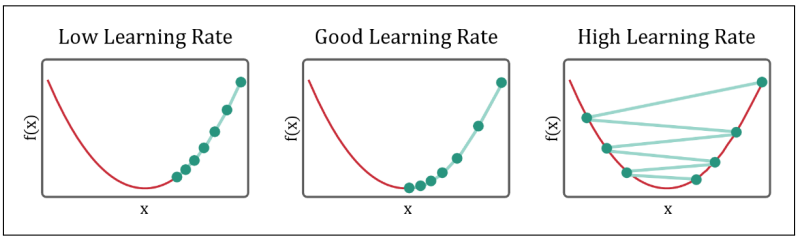

Abbildung 2.12: Konvergenzverhalten im Gradientenverfahren [Zhang et al., 2020, S. 429].

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

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

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

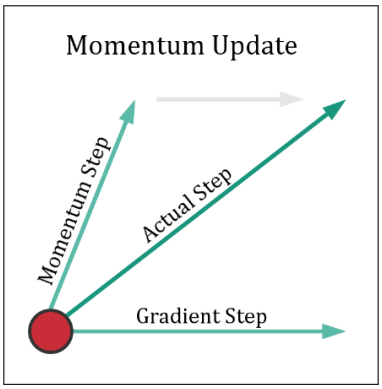

Abbildung 2.13: Gradientenverfahren unter Einfluss eines Momentums [CS231N, O. J.].

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

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

2.4 Transfer Learning

TL ist in den letzten Jahren wissenschaftlich immer bedeutsamer geworden, da DL-Modelle heutzutage sehr komplex und Trainingsprozesse sehr zeit- und rechenintensiv sind. Unter TL versteht man das Wiederverwenden bereits vortrainierter neuronaler Netze für die Lösung neuartiger Probleme. Das initiale Training obliegt hierbei meist großen Unternehmen oder Institutionen. Dabei werden die erprobten Modelle sodann als Startpunkt genutzt und nur noch auf die neuen Probleme adaptiert, anstatt eigene Modelle von Grund auf neu zu trainieren. Anwender profitieren hier zeitlich, qualitativ

und technisch. Zumeist sind architektonische Anpassungen in den hinteren Schichten der vortrainierten Modelle erforderlich, sodass sie sich für die Lösung der neuen Probleme eignen, wie Abbildung 2.14 veranschaulicht. Zudem ist ein gezieltes weitergehendes Training mit entsprechenden Daten notwendig. Inwieweit die neuen Daten auf die vortrainierten Modelle einwirken sollen, ist individuell zu erproben [Zhang et al., 2020, S. 554].

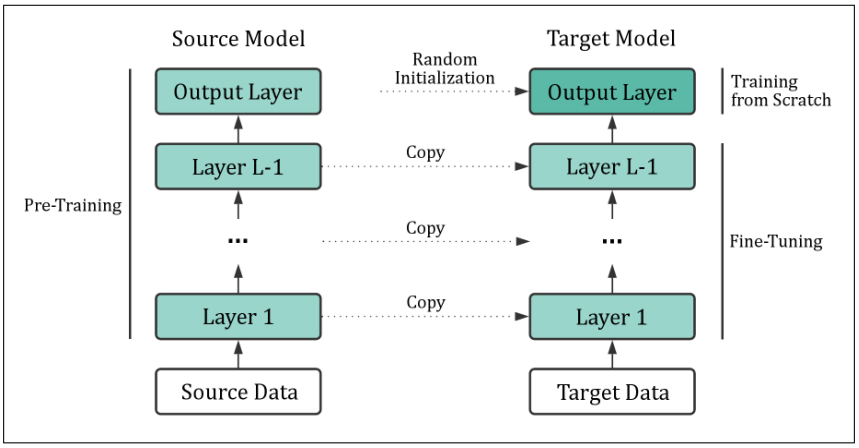

Abbildung 2.14: Fine-Tuning vortrainierter Modelle [Zhang et al., 2020, S. 555].

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

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

3 Natural Language Processing

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

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

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

3.1 Vorverarbeitung

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

3.1.1 Textbereinigung

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

3.1.2 Textnormalisierung

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

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

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

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

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

3.1.3 Tokenisierung

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

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

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

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

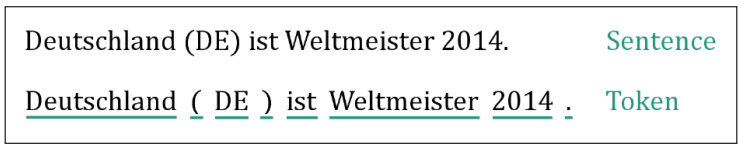

Abbildung 3.1: Tokenisierung eines beispielhaften Satzes.

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

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

3.2 Word Embeddings

Algorithmen können Texte bekanntermaßen nicht in ihrer Rohform verarbeiten. Texte bedürfen einer geeigneten mathematischen Form. Word Embeddings überführen Texte 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.

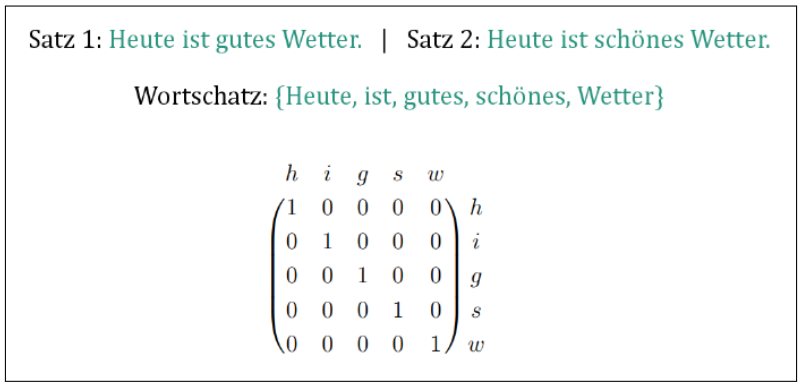

Abbildung 3.2: One-Hot-Encoding mit zwei beispielhaften Sätzen.

3.2.2 Bag-of-Words

Bag-of-Words (BOW) realisieren vornehmlich 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. 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	**Vögel**	**sind**	**Tiere**	**der**	**Lüfte**
Text 1	2	8	1	5	1
Text 2	1	3	0	2	0
Text 3	0	4	0	7	0

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

Skip-Gram-Modelle 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 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. Demnach kann der Kontext 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) 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].

3.2.5 Byte-Pair-Encoding

Text

3.2.6 GloVe

Text

-¿ Deep Language Models, Language Modeling, gegenüber normalen Word Embeddings relevant für die ATS

-¿ Word Embeddings in den Kontext des Language Modeling, d.h. Language Models einordnen (siehe Leveraging ...), ggf. bei den Deep Language Representations nach deren Definition aufgreifen

-¿ Keyword Network generieren, bestenfalls mit eigenem Modell, welches auf den eigenen Korpora basiert

-¿ Beschreibung im Kapitel der Encoder-Decoder-Architekturen unbedingt mit den jetzt neu gewonnenen Informationen aktualisieren

3.3 Deep Language Representations

Text

3.3.1 BERT

Text

3.3.2 ELMo

Text

3.3.3 GPT

Text

Notizen:

- BERT als Encoder & Decoder nutzen, Architekturen und TL dementsprechend aufgreifen (S. 1 in YAN19), bspw. als Encoder oder/ und Decoder verwenden, siehe ROT20 auf S. 2 rechts und S. 6 unten, Encoder zur NLU und Decoder zur NLG, d.h. BERT oder andere Transformer als vortrainiertes multilinguales Modell für Encoder/ Decoder nutzen, Ausblick auf Adaption von EN-¿DE: Fine-Tuning der Modelle, verschiedene Experimente, aber dazu im späteren Kapitel mehr, BERT ist außerdem austauschbar, durch sowohl größere als auch kleinere Modelle, Multilingualität architektonisch ergründen, erwähnen, dass diese Modelle die Encoder oder auch Decoder ersetzen können, hierzu populäre Ansätze wie [Rothe et al., 2020], Trainingsvorgehensweise von BERT und ELMo beschreiben, wie im Forschungsstand bereits erwähnt, konnten diese vortrainierten Language Models die NLU-Welt revolutionieren, wie viele Parameter wurden genutzt? Verschiedene Modelle vergleichen, bspw. das Notebook `https://colab.research.google.com/drive/`

1WIk2bxglElfZewOHboPFNj8H44_VAyKE?usp=sharing#scrollTo=4M2uzGLV9a_O unter Änalysis"referenzieren, jeweils maximale Token-Länge oder verschieden trainierte Versionen hervorheben, ggf. LongFormer als Encoder benutzen

- BERT vs. Alternativen: https://towardsdatascience.com/bert-roberta-distilbert-xlnet
- ELMo
- GPT
- Transfer Learning mit BERT hier sinnvoll, sodass das Modell die Sprache nicht in einer bestimmten Domain oder mit zu wenigen Texten neu erlernen muss, TL durch Encoder, Decoder etc. auf jeden Fall hier aufgreifen
- BERT zunächst in Englisch nutzen, weil SOTA, ggf. Grafik aus Oli's VL integrieren, irgendwie in Abstractive Summarization Pipelines integrieren
- BERT ist auch multilingual, d.h. englisches Modell mit deutschem Fine-Tuning vermutlich sogar brauchbar
- Modell beschreiben, d.h. Datensätze und Parametrisierung, SOTA für verschiedene NLP-Tasks, von Kapazitäten profitieren, hat viel Kontextwissen, Fine-Tuning für eigenes Problem, d.h. domainspezifisch o.ä.
- Am besten direkt ein vortrainiertes Transformer-Modell nutzen (extra für Summarization-Tasks), BERT und RL bspw. in der Pipeline integrieren, Ziel wäre dann: Verbesserung im Score erzielen
- BERT vielleicht durch andere (teils bessere und neuere) Transformer ersetzen? Transformer in NLP recherchieren, LSTM als veraltet bezeichnen

Notizen:

- Quelle: [Nitsche, 2019]
- Transfer Learning with German BERT? https://deepset.ai/german-bert -¿ Modell muss die deutsche Sprache nicht alleine und von neu mit den Trainingsdaten lernen, sondern erhält einen großen Vorsprung, BERT ist Modell, welches der Transformer-Architektur nachkommt, d.h. Transformer sind bestimmte Architekturen, eventuell hiermit die Struktur dieses Kapitels überarbeiten, hier für vor allem aus meinem privaten Verzeichnis das Paper "Pre-Training of Deep Bidirectional Transformers for Language Understanding using BERT"nutzen

- GLoVe-Embeddings nutzen, weil TF-IDF etc. nicht den Kontext eines Satzes betrachten
- Supervised Learning nutzen, aber es ist eventuell nicht genug, hier kommt bspw. Transfer Learning mit BERT zur Abhilfe, zudem bspw. semi-supervised Learning mit Auto-Encoders? Self-supervised Training
- Siehe Abstract im Exposé

4 Datengrundlage

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

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

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

Unter Berücksichtigung obiger Anforderungen werden nun drei Korpora ausgewählt. Der erste Korpus 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 aus-

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

Die anderen beiden Korpora dienen dem weitergehenden Training hinsichtlich der sprachtechnischen Adaption und bestehen demzufolge aus deutschsprachigen Datensätzen. Der erste Korpus hiervon wurde 2019 im Kontext der Swiss Text Analytics Conference als Grundlage eines Wettbewerbes publiziert und umfasst 100.000 Datensätze [Cieliebak, 2019]. Die Textinhalte entstammen der deutschsprachigen Wikipedia, weshalb auch hier von einer vielfältigen Domäne auszugehen ist. Letzteres gilt auch für den zweiten Korpus, welcher durch einen in Python selbst entwickelten Crawler generiert wurde. In einer Zeitspanne von sechs Monaten wurden mehr als 50.000 Nachrichtenartikel automatisiert kollektiert [Bird et al., 2009, S. 79, 83, 416]. Nach Sichtung der verfügbaren Daten können Artikel der ZEIT ONLINE als geeignet bewertet werden. Demnach sind etwa 15.000 Datensätze nutzbar. Texte dieser beiden Korpora bestehen durchschnittlich aus etwa 4.000 Wörtern, Zusammenfassungen hingegen aus etwa 250 Wörtern. Üblicherweise existiert beim anvisierten Training die Gefahr der sogenannten Exploitation. Diese Gefahr meint im Kontext der ATS konkret, dass das zugrundeliegende Modell die Struktur der Artikel anstatt der Inhalte der Artikel lernt. Grund für diese Annahme ist der typische Aufbau von Wikipedia-Artikeln. Diese beinhalten zumeist bereits im ersten Absatz stark verdichtete Informationen, also eine Art Zusammenfassung. Daher wird zur Vorbeugung eine Mischung aus den beiden deutschen Korpora vorgenommen [Bird et al., 2009, S. 42].

5 Abstraktiver Ansatz

Unter Kenntnis der zu implementierenden Architekturen, entsprechender NLP-Grundlagen und der Datengrundlage kann nun eine gesamtheitliche Architektur für die ATS konzipiert und trainiert werden. Dafür sind zunächst einige Vorbemerkungen zu spezifizieren.

Die Textinhalte aller Korpora bedürfen 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 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 sind somit messbar.

... Architektur beschreiben, auf bereits beschriebene Inhalte eingehen, Notizen beachten, Tokenlänge o.ä. bemerken, Quellen!

TODO: Leveraging-Paper hier umfänglich beschreiben, auf dokumentierte Inhalte dieser Arbeit eingehen + [Vaswani et al., 2017]

Notizen:

- Quelle: [Nitsche, 2019]
- Abgrenzung zum extraktiven Ansatz beschreiben
- Vorteile gegenüber referenzierten Modellen herausstellen
- Generierung neuer Sätze sowohl mit vorkommenden als auch mit nicht-vorkommenden Wörtern (vgl. Paper: „Automatic Text Summarization with Machine Learning“)

- Extraktive Zusammenfassung: Potenzielle Features bzw. Kennzahlen: Übereinstimmung mit dem Titel, Satzposition, Satzähnlichkeit, Satzlänge, domänenspezifische Wörter, Eigennamen, numerische Daten

5.1 Architektur

Notizen:

- Eignung von Python für ML/ DL
- Anmerkungen zum CNN-Korpus aus der Datengrundlage aufgreifen und mit Colab-Notebook vervollständigen, d.h. wie viele Texte sind länger als BERT-MAX 512 Token? Was haben Experimente ergeben? Token-Länge/ Longformer? Gibt es alternative Lösungen? Z.B. Longformer -¿ Ausblick
- Multilingual BERT: `https://huggingface.co/bert-base-multilingual-cased`
- `https://colab.research.google.com/drive/1WIk2bxglElfZewOHboPFNj8H44_VAyKE?usp=sharing#scrollTo=ZwQIEhKOrJpl`, `https://huggingface.co/transformers/training.html#trainer`, `https://www.quora.com/What-is-the-difference-between-FP16-` `https://margaretmz.medium.com/setting-up-aws-ec2-for-running-jupyter-notebook-o` + Jupyter Notebook auf Tesla 80K `https://www.nvidia.com/de-de/data-center/tesla-k80/`
- [Rothe et al., 2020] um Encoder und Decoder zu ersetzen, laut beschriebener Encoder-Decoder-Architektur + Language Model + Transfer Learning in Kombination, also die drei Grundlagenkapitel werden hier zusammengeführt, sogenannter Warm-Start des Encoder-Decoder-Models, d.h. kein initiales Training erforderlich, welches im Bereich von NLP-Tasks ein Neuerlernen einer Sprache bedeuten würde, Architektur skizzieren, in der sowohl Encoder als auch Decoder "warm gestartet"werden, Fine-Tuning auch skizzieren, Warm-Starting beschreiben, wie hier in der Theorie beschrieben: `https://colab.research.google.com/drive/1WIk2bxglElfZewOHboPFNj8H44_VAyKE?usp=sharing#scrollTo=Gw3IZYrfKl4Z`, Skizzen von dort nutzen, Encoder-Decoder-Weight-Sharing beschreiben, zuvor muss hier erstmal beschrieben werden, wie die Encoder-Decoder-Architektur konkret aussieht, also BERT o.ä., dann verschiedene Skizzen und Dinge wie Weight-Sharing beschreiben, prinzipiell das Colab-Notebook nutzen, Multilingualität behandeln
- [Yang et al., 2019] S. 4 rechts, S. 5 oben für Evaluation, S. 6 links unten für Konfiguration

- Frage: Fine-Tuning auf CNN/ Dailymail für Funktionalität der ATS auf englischer Sprache, d.h. Modell soll auf ATS fokussiert werden (bspw. von HuggingFace übernehmen), dann weiteres Fine-Tuning auf deutschen Korpus zur sprachlichen Adaption, also zweistufig?
- Modellauswahl begründen, d.h. warum "Transformer"? Warum genau dieses vortrainierte Modell? Auf vorherige Inhalte der Masterarbeit verweisen
- Transformer-to-Transformer-Architektur beschreiben, Grundlagen des TL, der Encoder/ Decoder und der Embeddings/ Language Model Representations etc. aufgreifen, bspw. Bert2Bert, Longformer2Roberta wie in `https://huggingface.co/patrickvonplaten/longformer2roberta-cnn_dailymail-fp16/blob/main/README.md`
- Netzwerk des abstraktiven Ansatzes als Pipeline skizzieren
- Vorgehen bei der Datenverarbeitung/ -vorbereitung wie im Notebook beschreiben, d.h. warum cased/ uncased? Token-Länge von BERT o.ä. als Begründung erwähnen
- Transformers-Library -¿ Scores in Excel, funktioniert gut als Benchmark/ "Nullfall"
- Komponenten wie den Encoder oder den Decoder mit der behandelten Theorie auf die durch die Datengrundlage gestellten Anforderungen mappen, d.h. tatsächlich eine Auswahl aus den vorgestellten theoretischen Inhalten treffen und stets ausführlich begründen
- Seq2Seq-Trainer wie im Notebook beschreiben, Notebook nochmal durchlesen und wichtige Informationen sinnvoll einbinden
- Seq2Seq `https://github.com/yaserkl/RLSeq2Seq#dataset` -¿ Fehler
- Seq2Seq-Library: `https://github.com/dongjun-Lee/text-summarization-tensorflow` -¿ Done, aber Scores auf meinem Datensatz nicht evaluierbar, da Aufbau des Vocabularies und Training auf meinem Korpus ausstehend ist, aber zu rechenintensiv ist, alternativ nur Scores auf anderem Korpus auswertbar, Azure ML vs. AWS SageMaker?
- Deep Reinforcement Learning (DeepRL) for Abstractive Text Summarization `https://medium.com/analytics-vidhya/deep-reinforcement-learning-deeprl-for-abstractive`

-¿ Rouge-Scores ausschließlich auf dem CNN-Korpus berechnet, Anpassungen an den Daten, an der Code-Architektur und an den Modellen möglich -¿ Zurückgestellt aber bei Bedarf mit Potenzial

- BERT-Encoder Transformer-Decoder: Paper `https://arxiv.org/pdf/2008.09676.pdf`, Code `https://github.com/nlpyang/PreSumm`, Results `https://paperswithcode.com/paper/abstractive-summarization-of-spoken#code` -¿ In Progress...
- Deep Reinforced Model with PyTorch: `https://github.com/rohithreddy024/Text-Summarizer-Pytorch` -¿ TBD

5.2 Training

Notizen:

- ZIH-Cluster mit HPC/ Taurus aus IT- /ML-Infrastruktur zum Training, Spezifikationen etwas beschreiben
- Setup beschreiben `https://docs.aws.amazon.com/dlami/latest/devguide/tutorials.html`, `https://youtu.be/pK-LYoRwp-k?list=WL`
- Konfiguration
- Training verschiedener Modelle
- Kompressionsrate der Referenzzusammenfassungen in Bezug auf die Originaltexte liegt bei 12 Prozent, d.h. mit einer gewissen Toleranz wird die maximale Zusammenfassungslänge auf 15 Prozent des Originaltextes festgelegt
- Architektur beschreiben, d.h. IT-Infrastruktur + GPU-Architektur

5.3 Evaluation

Notizen:

- Zwecks Einleitung für ROUGE auf die Einleitung im NLP-Kapitel eingehen, d.h. Mehrdeutigkeit der Lösungen bzw. mehrere korrekte Lösungen, Metriken zur Evaluation dieser Korrektheit erforderlich, verschiedene Ansätze nachfolgend

- ROUGE vorstellen, evtl. BLEU, auch die Implementierung beschreiben, ROUGE-Score umstritten, daher evtl. alternativen Score nutzen, Kritik inkl. Grenzen nennen, dennoch weitreichend genutzt und daher vergleichbar, wichtig für die Interpretation, ROUGE evaluiert uncased, `https://huggingface.co/metrics/rouge`
- Kompressionsrate messen
- Accuracy, Precision & Recall inkl. Trade-Off auf S. 239 BIR09
- Qualität der Zusammenfassung messen (BLEU `https://en.wikipedia.org/wiki/BLEU`, ROUGE `https://en.wikipedia.org/wiki/ROUGE_(metric)`, evtl. Funktionen fusionieren)
- Evaluation verschiedener Modelle mit geeigneter Vergleichstabelle
- Vergleich mit SOTA-Modellen
- Praktische Nutzung durch Implementation eines vortrainierten Modells in ein Skript oder eine Software
- Es muss eine Metrik existieren, mit der man die Genauigkeit bzw. Qualität der Zusammenfassung messen kann, d.h. man möchte die Texte nicht mit menschlich generierten Zusammenfassungen vergleichen, sondern automatisiert lernen, ggf. sollte man auch Grammatik und Inhalt separat prüfen
- For a given document there is no summary which is objectively the best. As a general rule, many of them that would be judged equally good by a human. It is hard to define precisely what a good summary is and what score we should use for its evaluation. Good training data has long been scarce and expensive to collect. Human evaluation of a summary is subjective and involves judgments like style, coherence, completeness and readability. Unfortunately no score is currently known which is both easy to compute and faithful to human judgment. The ROUGE score [6] is the best we have but it has obvious shortcomings as we shall see. ROUGE simply counts the number of words, or n-grams, that are common to the summary produced by a machine and a reference summary written by a human. `https://towardsdatascience.com/deep-learning-models-for-automatic-summarization-4c2b89f2`
- Bei der Anwendung einer Architektur, in der das Modell durch Reinforcement Learning trainiert wird, braucht man keine massenhaft menschlich generierten Referenztexte, sondern eine wohlbedachte Kostenfunktion, der ein entsprechender

Aufwand entgegen gebracht werden muss, d.h. die Herausforderung liegt beim RL eher darin, eine Umwelt und eine geeignete Funktion zum Belohnen und Bestrafen zu konstruieren, hier sind bspw. auch Evaluationsmetriken notwendig

- Rouge-Score in Python: `https://pypi.org/project/rouge-score/`
- Typisches Diagramm zur Visualisierung des Trainingsprozesses anfügen
- Verhalten des Modells interpretieren und Anpassungen ableiten, bspw. Exploitation wegen der Struktur der Artikel nochmal aufgreifen

6 Sprachtechnische Adaption

- Konzeption der Adaption
- Adaption von EN-¿DE: Fine-Tuning der Modelle, verschiedene Experimente, umfangreiche Tabellen aufstellen

6.1 Konzeption

Text...

6.2 Training

Text...

6.3 Evaluation

Text...

7 Zusammenfassung

Notizen:

- Methoden und Ergebnisse zusammenfassen
- Bewertung der Zielerreichung
- Beantwortung der Forschungsfragen
- Lösung eingangs beschriebener Szenarien
- Welche Domäne wird am besten erkannt? Funktionieren Modelle mit gemischten Korpora?
- Siehe Abstract im Exposé

8 Diskussion und Ausblick

Notizen:

- Adaptive Learning für die Modelle ansatzweise vorstellen
- Modelle für mehrere Sprachen trainieren
- Modell auf Dialogcharakter adaptieren, um es in der Verdichtung von Protokollen einer Videosprechstunde zu nutzen, bzw. generell bspw. Meetings zusammenzufassen
- Forschungsstand und SOTA-Modelle hierfür im einleitenden Kapitel beschreiben, hier aufgreifen (vgl. Paper: „Abstractive Dialogue Summarization with Sentence-Gated Modeling Optimized by Dialogue Acts“ und “Using a KG-Copy Network for Non-Goal Oriented Dialogues”), bereits Architekturen vorstellen (vgl. „Automatic Dialogue Summary Generation of Customer Service“ und „Dialogue Response Generation using Neural Networks and Background Knowledge“ und „Global Summarization of Medical Dialogue by Exploiting Local Structures”)
- Modell ohne Anpassungen auf Konversationen anwenden: `https://www.isi.edu/natural-language/people/hovy/papers/05ACL-email_thread_summ.pdf`
- Modell nutzen, um Zusammenfassungen für Texte zu generieren und damit neue Datensätze für neue Modelle zu generieren, aber stark von der Qualität abhängig
- Gelb markierte Literatur sichten und verwenden, Datumsangaben aktualisieren
- Ausblick: Zusammenfassungen formatieren, also Großschreibung nach Satzenden oder auch Leerzeichenentfernung vor Punkten, Adaption auf Sprache und/ oder Domain
- Ausblick: Ausblick: Datengrundlage besteht aus frei verfügbaren allgemeinsprachlichen, ausreichend langen und deutschsprachigen Daten, verschiedene Herkünfte, auf Grundlage dieser Allgemeinsprache und den eben genannten Vorhaben, sollte

ein grundlegendes Modell trainiert werden und später für den Use Case eine Art Adaptive Learning betrieben werden, d.h. wenn bekannt ist, dass das Modell für medizinische Texte angewandt werden soll, sollte man vorher die Parameter des Modells finetunen, Gefahr: Initial bereits hohe Informationsdichte bei Diktaten - "Was fällt raus?"

- Ausblick: Limitations von NLP: [Bird et al., 2009, S. 30-31]
- Siehe Abstract im Exposé

Literaturverzeichnis

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[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.

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

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

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

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

[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.

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

Thesen

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

Selbstständigkeitserklärung

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

Dresden, den ???. Juli 2021

Daniel Vogel

A Erster Anhang

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

B Zweiter Anhang

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