INSTITUT FÜR INFORMATIK
Datenbanken und Informationssysteme

Universitätsstr. 1 D–40225 Düsseldorf

Heinrich Heine Universität Düsseldorf

# Generation von Multi-Document Summarizations mittels Variational Auto Encoder Transformern

**Lionel Schockenhoff**

Masterarbeit

| | |
|---|---|
| Beginn der Arbeit: | 20. August 2021 |
| Abgabe der Arbeit: | 31. September 2010 |
| Gutachter: | Prof. Dr. Stefan Conrad |
| | Prof. Dr. Gunnar Klau |

## Erklärung

Hiermit versichere ich, dass ich diese Masterarbeit selbstständig verfasst habe. Ich habe dazu keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Düsseldorf, den 31. September 2010

_____

Lionel Schockenhoff

## Zusammenfassung

Hier kommt eine ca. einseitige Zusammenfassung der Arbeit rein.

# Inhaltsverzeichnis

# 1 Einleitung

Das automatisierte Zusammenfassen von Dokumenten ist im Bereich des Natural Language Processing (NLP) eine große Herausforderung. Natural Language Processing ist ein Unterbereich der künstlichen Intelligenz, der sich mit dem maschinellen Verarbeiten von natürlicher Sprache auseinandersetzt. Aufgrund von neuen Methoden und immer größeren, komplexeren Netzwerken lassen sich hervorragende Ergebnisse in diversen NLP Aufgabenbereichen erzielen. Insbesondere lassen sich große, kostenspielig vortrainierte Modelle mit vielen Parametern durch Transfer Learning in diversen speziellen Aufgabenbereichen einsetzen.

GPT-2 erzielte unter Verwendung von Transformern großartige Ergebnisse bei der Textgenerierung, unter anderem auch bei der abstraktiven Zusammenfassung von Texten. Die abstraktive Textzusammenfassung bezeichnet das Zusammenfassen von Texten zu einem kurzen, präzisen Text.

## 1.1 Motivation

Im Bereich des E-Commerce und von Online-Vergleichsportalen entstehen eine große Anzahl an Rezensionen zu unterschiedlichen Produkten, Dienstleistungen und Anbietern. Es ist für den Anwender eine Herausforderung sich einen repräsentativen Überblick über die einzelnen Rezensionen zu verschaffen. Ein gängiges Bewertungssystem ist das Berechnen des arithmetischen Mittels über die einzelnen Scores aller Bewertungen. Ein zusammengefasster Bewertungsscore ist nicht ausreichend repräsentativ für viele Produkte und beleuchtet nicht spezifische Funktionen der einzelnen Produkte. Hier bietet es sich an, ein Produkt durch eine generierte Rezension zu repräsentieren, die die anderen Rezensionen repräsentativ zusammenfasst und auf die verschiedenen wichtigen Aspekte der Produkte eingeht. So kann ein Anwender sich zeitsparend einen allumfassenden Überblick über bestimmte Produkte verschaffen.

## 1.2 Ziel und Aufbau der Arbeit

Ziel dieser Masterarbeit ist das unbeaufsichtigte abstraktive Zusammenfassen von mehreren Produktrezensionen mittels Variational Autoencodern zu einer repräsentativen Rezension. Die generierten Zusammenfassungen sollten möglichst viele Aspekte der ursprünglichen Rezension aufgreifen und einen guten Gesamtüberblick über die Rezensionen zu einem Produkt oder Restaurant geben. Als Datengrundlage gelten hier der Amazon-Review- und Yelp-Review-Datensatz, die zu Produkten und Restaurants mehrere unterschiedliche diverse Rezensionen liefern.

Mittels zwei unterschiedlichen Variational Autoencodern Optimus und BiMeanVAE werden die Zusammenfassungen generiert. Optimus basiert auf den beiden bekannten Sprachmodellen BERT und GPT-2. BiMeanVAE besteht aus einem bidirektionalem LSTM-Encoder und einem LSTM-Decoder. Des Weiteren werden die verwendeten Variational Autoencoder durch ein Attribut-Modell optimiert, um präzisere Rezensionen zu generieren.

Abschließend werden die vorgestellten Variational Autoencoder mit den entsprechenden Optimierungen auf den beiden Datensätzen evaluiert und untereinander verglichen.

## 1.3 Terminologie

**Adaptive Moment Estimation** (Adam) ist ein Optimierer für neuronale Netze.

**Fine-Tuning** ist ein Trainingsprozess, der ein bereits vortrainiertes neuronales Netz an eine spezielle Aufgabenstellung anpasst und auf diese trainiert. Die bereits im Pre-Training gefundenen Parameter werden weiter angepasst.

**Global Vectors for Word Representation** (GloVe) ein Verfahren für das Zuweisen von Vektorrepräsentationen zu Worten.

**JavaScript Object Notation** (JSON) ist ein in Textform vorliegendes Datenformat.

**Natural Language Processing** (NLP) beschreibt das maschinelle Verarbeiten und Analysieren von natürlichen Sprachen unter Verwendung von unterschiedlichen Techniken und Verfahren.

**Pre-Training** beschreibt den initialen Trainingsprozess eines neuronalen Netzes. Im Bereich des NLP wird Pre-Training verwendet, um ein allgemeines Sprachmodell zu trainieren, welches später auf individuelle Aufgabenstellungen nachtrainiert wird.

**Tokenisierung** ist die Segmentierung eines Textes in eine Folge von Tokens. Die einzelnen Tokens sind Zeichenketten bestehend aus einem oder mehreren Zeichen.

# 2 Grundlagen

In diesem Kapitel werden die Grundlagen zu den verwendeten Technologien erklärt. Zunächst wird die in dieser Masterarbeit untersuchte Aufgabe des abstrakten Zusammenfassens von mehreren Dokumenten erläutert. Anschließend werden die Grundlagen zu neuronalen Netzen und insbesondere zu Deep Learning erklärt. Hier wird ebenfalls die Struktur von Sequence-To-Sequence Modellen und LSTM-Zellen erläutert.

Im folgenden Kapitel 3 wird anschließend aus den Grundlagen die verwendete Transformer Architektur, BERT und GPT-2 erklärt.

## 2.1 Textzusammenfassung

Das automatisierte Zusammenfassen von Texten ist ein Teilgebiet des NLP, welches sich mit dem Zusammenfassen von langen Texten zu einem kongruenten, kürzeren Text unter Beibehaltung von wichtigen Informationen befasst. Durch die zunehmenden Datenmengen wird automatisierte Textzusammenfassung immer relevanter, um akkurate Zusammenfassungen und Überblicke zu geben. Automatisierte Textzusammenfassung lässt sich zum Beispiel bei der Generierung von Kurzzusammenfassungen zu Dokumenten verwenden.

Grundsätzlich wird beim automatisierten Zusammenfassen von Texten zwischen den extraktiven und den abstraktiven Methoden unterschieden.

### 2.1.1 Extraktive Textzusammenfassung

Extraktive Textzusammenfassung ist das Identifizieren und anschließende Extrahieren von wichtigen Phrasen oder Sätzen aus dem Ursprungstext. Hierbei werden die entsprechend ausgewählten Phrasen in der Zusammenfassung genauso wie sie im Ursprungstext vorkommen übernommen. Die zu extrahierenden Phrasen oder Sätze werden mithilfe einer Scoringfunktion gefunden und später aneinander gereiht. Hierzu gibt es unterschiedliche Methoden und Metriken.

### 2.1.2 Abstraktive Textzusammenfassung

Abstraktive Textzusammenfassung versucht durch Interpretation und Verständnis des Ursprungtexts eine kurze, kongruente Zusammenfassung zu reproduzieren. Die Zusammenfassung soll alle wichtigen Informationen enthalten und als zusammenhängender flüssiger Text erzeugt werden. Ein kongruenter Text wird erzeugt, da das Sprachmodell frei Sätze produzieren kann und nicht an vorher vorgegebene Sätze oder Phrasen gebunden ist, wie bei der extraktiven Zusammenfassung.

Große Fortschritte im Bereich der abstraktiven Textzusammenfassung ergaben sich in den letzten Jahren durch Sequence-To-Sequence Modelle. Diese encodieren den Eingabetext in eine Übergangsrepresentation und generieren aus dieser durch Decodierung eine Ausgangsrepresentation.

### 2.1.3 Multi-Document Textzusammenfassung

Eine große Herausforderung ist das Zusammenfassen von mehreren Dokumenten über dasselbe Thema zu einem einzigen Dokument. Die entstehende Zusammenfassung soll Anwendern einen guten und schnellen Überblick über eine große Anzahl an Dokumenten bieten. Die unterschiedlichen Dokumente enthalten diverse Informationen die nicht immer deckungsgleich sind. Somit ergibt sich die Herausforderung, unterschiedliche Perspektiven in den jeweiligen Dokumenten zusammenfassend zu representieren. Da sich unterschiedliche Standpunkte schlecht mittels extraktiven Methoden darstellen lassen, bieten sich bei der Multi-Document Textzusammenfassung abstraktive Methoden an. Somit kann eine kongruente Zusammenfassung generiert werden, die die unterschiedlichen Aspekte der Dokumente darstellt.

In dieser Masterarbeit werden unterschiedliche Bewertungen zu Produkten und Restaurants zusammengefasst. Insbesondere Bewertungen unterscheiden sich stark in ihren Standpunkten und können positiv, negativ oder neutral sein und auf sehr spezifische Eigenschaften der entsprechenden Produkte eingehen. Es ist eine große Herausforderung aus einer Menge aus Produktbewertungen eine allgemeingültige zusammenfassende Bewertung zu produzieren, die klar strukturiert, kongruent und gut lesbar die entsprechenden Inhalte wiedergibt.

## 2.2 Deep Learning

Deep Learning ist ein Teilbereich des Machine Learning, bei dem nach dem Vorbild für das menschliche Gehirn neuronale Netze verwendet werden. Neuronale Netze werden unter anderem beim Natural Language Processing eingesetzt. Die neuronalen Netze bestehen aus mehreren Layern, die sequentiell den Output des vorherigen Layers weiterverarbeiten. Das Ziel von Deep Learning ist, durch Training der neuronalen Netze, Repräsentationen beziehungsweise Approximationen für Funktionen in den Daten zu finden. Zum Trainieren dieser Netze wird oft Backpropagation, ein Verfahren zur Berechnung der Gradienten in neuronalen Netzen und entsprechender Anpassung der Gewichte verwendet.

## 2.3 Word Embeddings

Word Embeddings werden im Natural Language Processing verwendet, um Wörter mit aussagekräftigen Vektoren zu repräsentieren. Hochdimensionale Objekte wie Wörter lassen sich mittels Word Embeddings in einen niedrigdimensionalen Raum einbetten und behalten dabei ihre semantischen Relationen bei. Auf diesen Vektoren lassen sich unterschiedliche arithmetische Operationen ausführen.

Bekannte kontextunabhängige Word Embedding Verfahren sind zum Beispiel word2vec (Mikolov et al., 2013) und GloVe (Global Vectors for Word Representation) (Pennington et al., 2014). Diese Verfahren berechnen für jedes Wort einen universellen Vektor, der alle unterschiedlichen Features dieses Wortes enthält. Die errechneten Vektoren sind kontextunabhängig und für ein Wort wird stets der gleiche Vektor verwendet. Word2vec erlernt die entsprechenden Vektorrepräsentationen mittels eines SkipGram neuronalen Netzes

(Mikolov et al., 2013) GloVe hingegen über die nicht Nulleinträge einer Co-occurrence Matrix von Wörtern untereinander (Pennington et al., 2014).

Kontextabhängige Verfahren, wie zum Beispiel ELMO (Peters et al., 2018) oder BERT (Devlin et al., 2018), generieren kontextabhängige Vektoren für Wörter und Berücksichtigen dabei das Umfeld in dem das einzelne Wort auftritt. Bei diesen Verfahren wird zur Bestimmung eines Word Embeddings der gesamte Satz benötigt, um die erlernten kontextspezifischen Eigenarten für die Einbettung zu berücksichtigen.

Verfahren wie zum Beispiel BERT nutzen besondere Tokenisierungsmethoden, die es erlauben einzelne Wörter durch mehrere Tokens zu repräsentieren. Dieses Splittingverfahren von Wörtern in Subtokens ist als WordPiece Verfahren (Wu et al., 2016) für BERT und als BytePairEncoding (Sennrich et al., 2015) für GPT-2 bekannt. Durch das Splitten in Subtokens und Erlernen von Einbettungen für diese lässt sich ein kleineres Wörterbuch erstellen. Da sich die Wörter stets in Subtokens zerlegen lassen können Out-of-Vocabulary-Fehler vermieden werden.

## 2.4 Sequence-To-Sequence Modelle

Sequence-To-Sequence Modelle sind eine Gruppe von Deep Leaning Modellen zur Sprachverarbeitung die eine Textsequenz $(x_1, x_2, \ldots, x_n)$ in eine andere Textsequenz $(y_1, y_2, \ldots, y_n)$ überführen (Sutskever et al., 2014). Die Besonderheit bei diesem Modell liegt darin, dass die Länge der Eingabesequenz und der Ausgabesequenz unterschiedlich sein kann. Die am meisten verwendeten Sequence-To-Sequence Modelle bestehen aus einer Encoder/Decoder Architektur. Der Encoder und Decoder besteht jeweils aus sequentiellen Rekurrenten Neuronalen Netzen, die jeweils als Eingabe ein Token und den HiddenState des vorherigen RNNs erhalten. Somit summieren sich innerhalb des Encoders die HiddenStates auf und werden abschließend durch einen HiddenState Vektor repräsentiert, der die Informationen aus der Eingabe enthält. Der abschließende HiddenState Vektor lässt sich wie folgt iterativ bestimmen, wobei $f$ für die entsprechende RNN-Zelle und $W$ für die Gewichtsmatrizen steht:

$$h_t = f(W^{hh} h_{t-1} + W^{hx} x_t) \tag{1}$$

Oft werden als RNN-Zellen LSTM- oder GRU-Zellen verwendet. Der Decoder ist ebenfalls ein RNN, welcher aus dem HiddenState Vektor des Encoders eine Ausgabesequenz generiert.

Um die Performance von Sequence-To-Sequence Modellen weiter zu optimieren werden in Kapitel 3.2 Attention Modelle eingeführt.

## 2.5 Long Short Term Memory

**L**ong **S**hort **T**erm **M**emory (LSTM) Netze sind eine Untergruppe der Rekurrenten neuronalen Netze (RNN), die es ermöglichen Langzeitbeziehungen in Daten zu erlernen (Hochreiter und Schmidhuber, 1997). Rekurrente neuronale Netze zeichnen sich durch ihre rekurrenten Verbindungen innerhalb desselben Layers aus, wodurch die nächste

RNN-Zelle die vorherigen Informationen weiterverarbeiten kann. Ein häufiges Problem ist das Modellieren von Langzeitbeziehungen mittels RNNs. LSTMs sind speziell darauf ausgelegt diese Langzeitbeziehungen abzubilden, indem sie stets einen Zellzustand $c_t$ mit übergeben. Eine LSTM-Zelle hat nun die Möglichkeit diesen Zellstatus zu manipulieren, indem Informationen hinzugefügt oder gelöscht werden können. Diese Manipulationen des Zellstatus wird durch drei Gates ermöglicht. Erstens dem Forget Gate, welches der Zelle ermöglicht alte Werte zu vergessen. Zweitens dem Input Gate, welches steuert zu welchem Ausmaß neue Eingabewerte in dem HiddenState gespeichert werden. Drittens dem Output Gate, welches steuert welcher Teil des HiddenStates in die folgende Zelle propagiert wird.

Durch die Anpassung des HiddenStates in den einzelnen Zellen können relevante Informationen anhand des HiddenStates durch die Zellen geleitet werden, wodurch lange Beziehungen in den Textsequenzen modelliert werden können (Olah, o. D.). Das Propagieren durch eine Reihe von LSTM-Zellen, sowie der Einfluss der einzelnen Gates ist in Abbildung 2.5 dargestellt.

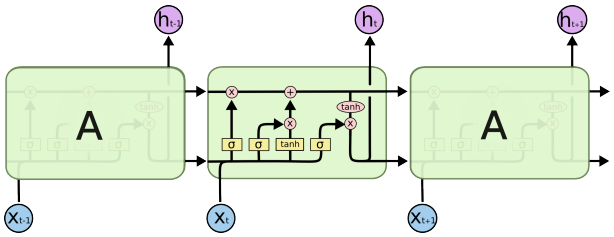



Abbildung 1: Reihe von LSTM-Zellen mit entsprechenden Gates (Olah, o. D.)

# 3 Transformer

Transformer sind eine spezifische Encoder-Decoder Deep Learning Architektur, die insbesondere im Natural Language Processing eingesetzt wird. Basierend auf Attention Layern ermöglichen Transformer die Leistung bei vielen NLP Anwendungen stark zu steigern. Ein großer Vorteil von Transformermodellen ist die Parallelisierbarkeit. Hierdurch lassen sich extrem große Sprachmodelle, wie zum Beispiel BERT (**B**idirectional **E**ncoder **R**epresentations from **T**ransformers) oder GPT-2 (**G**enerative **P**re-trained **T**ransformer-2) auf Basis von Transformern trainieren. Beide Modelle erzielen in den unterschiedlichsten Benchmarks hervorragende Ergebnisse.

Transformermodelle sind Sequence-To-Sequence Modelle, die aus einem Encoder und einem Decoder bestehen. Die vortrainierten Sprachmodelle BERT und GPT-2 lassen sich auf spezielle Aufgabenbereiche finetunen.

## 3.1 Transformer Encoder / Decoder

Transformer verwenden eine Encoder - Decoder Struktur. Der Encoder wandelt eine Eingabesequenz $(x_1, \ldots, x_n)$ in eine kontinuierliche Übergangsrepräsentation $(z_1, \ldots, z_n)$ um. Mit dieser Übergangsrepräsentation $z$ kann anschließlich vom Decoder eine Ausgabesequenz $(y_1, \ldots, y_n)$ generiert werden. Der Transformer Decoder generiert Textsequenzen autoregressiv und bezieht also für die Berechnung des nächsten Ausgabetokens die vorherigen Tokens als Eingabe mit ein.

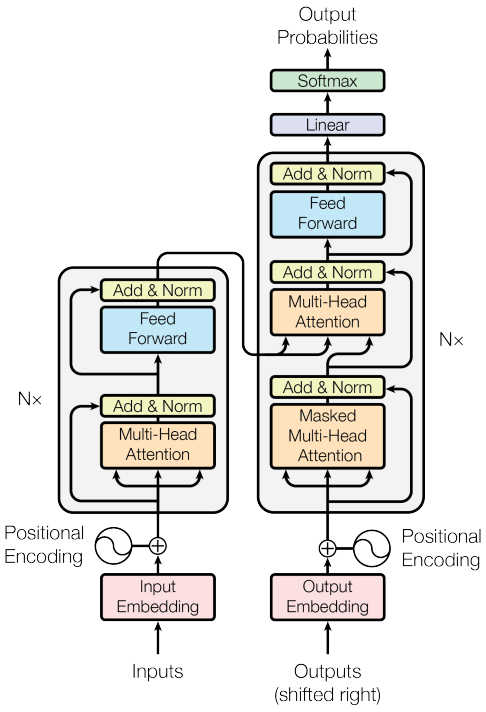



Abbildung 2: Transformer Encoder (links) und Transformer Decoder (rechts) (Vaswani et al., 2017)

Abbildung 3.1 stellt die Architektur und den Datenfluss durch die verschiedenen Layer von Transformer Encodern und Decodern dar. Grundsätzlich bestehen sowohl der Encoder sowie auch der Decoder aus aufeinanderfolgenden Multi-Head Attention-Layern gefolgt von Fully-Connected Feed-Forward-Netzen.

Die Transformer Encoder Blöcke verwenden Self-Attention-Layer und beziehen damit ihre Eingabe vollständig aus der Ausgabe des vorherigen Layers. Die meisten Modelle verwenden mehrere Transformer Encoder Blöcke und Decoder Blöcke sequentiell hintereinander.

Nach Encodierung der Eingabesequenzen durch den Transformer Encoder wird die Ausgabe in Attention Key- und Value-Vektoren umgewandelt. Diese Attentionvektoren werden vom Transformer Decoder in den Encoder-Decoder Attention Layern verwendet. Im Gegensatz zu den Self-Attention-Layern der Encoder verwendet der Decoder hier die Ausgabe des Encoders im Attention-Layer in den Berechnungen mit, um sich auf die korrekten Stellen im Input bei der Generierung der finalen Ausgabesequenz zu fokussieren.

## 3.2 Attention-Layer

Durch die Attention-Layer von Transformermodellen lassen sich relevante Informationen erkennen und das Modell sich auf diese fokussieren. Der Attention-Layer innerhalb des Transformers berechnet die Relevanz von Values zu bestimmten Keys und Queries.

Falls Attention-Layer ihre gesamte Eingabe wie bei Transformer Encodern aus einer Datenquelle beziehen sind diese Self-Attention-Layer. Bei Self-Attention-Layern werden die Query- $(Q)$, Key- $(K)$ und Value- $(V)$ Vektoren aus dem Eingabevektor $(X)$ durch Multiplikation mit erlernten Gewichtsmatrizen $(W)$ generiert (Vaswani et al., 2017).

$$Q = X \times W^Q, K = X \times W^K, V = X \times W^V \tag{2}$$

Durch ein Dot-Product zwischen Query- und Key-Vektoren der entsprechenden Eingabewörtern mit anschließender Softmax Normalisierung lässt sich ein Score errechnen, der den Fokus im Valuevektor auf das jeweilige Wort angibt (Vaswani et al., 2017).

$$Attention(Q, K, V) = Softmax(\frac{Q \times K^T}{\sqrt{d_k}}) \times V \tag{3}$$

In Gleichung 3 werden die Berechnungen an den Matrizen $Q$ für die Queries, $K$ für die Keys und $V$ für die entsprechenden Values durchgeführt. Als Skalierungsfaktor wird $\sqrt{dk}$ , die Dimension der Key Vektoren verwendet, um einen stabilen Gradienten zu erhalten.

Attention Funktionen lassen sich parallel ausführen. Diese Eigenschaft nutzen Transformermodelle aus und verwenden Multi-Head Attention Layer, um mehrere Attention Funktionen parallel auszuführen. Es werden parallel $h$ voneinander unabhängige Attention Layer ausgeführt, die ihre einzelnen Ergebnisse für die Weiterverarbeitung aneinanderfügen und linear in die gewünschte Dimension transformieren.

$$\begin{aligned} MultiHeadAttention(Q, K, V) &= [head_1, \ldots, head_h] \times W^O \\ \text{mit } head_i &= Attention(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{4}$$

In Gleichung 4 wird deutlich, dass die unterschiedlichen Attention-Heads voneinander unabhängig sind. Hierdurch können sich die einzelnen Attention-Heads auf unterschiedliche Merkmale in den Daten fokussieren.

## 3.3 Bidirectional Encoder Representations from Transformers

BERT (**B**idirectional **E**ncoder **R**epresentations from **T**ransformers) ist ein von Google (Devlin et al., 2018) entwickeltes Encoder-Sprachmodell, welches sich durch die bidirektionale kontextuelle Einbettung von Worten auszeichnet. Bei der Veröffentlichung von BERT konnten in vielen NLP Benchmarks Bestleistungen erzielt werden.

BERT zeichnet sich ebenfalls durch die Möglichkeit des Transfer Learnings aus. Es existieren vortrainierte BERT-Sprachmodelle, die jeweils auf spezifische Aufgaben finegetuned werden können.

Die Modellarchitektur von BERT sind sequentielle Transformerencoderlayer. Das Basis BERT-Modell verwendet 12 Transformerencoderblöcke mit jeweils 12 Multi-Attention-Heads und einer Hidden-Size von 768. BERT wird mit den beiden Pretrainings Aufgaben Masked Language Modeling und Next Sentence Prediction vortrainiert. Durch den besonderen Trainingsprozess betrachtet BERT Sequenzen von links und rechts gleichzeitig also bidirektional. Einige andere Verfahren betrachten Sequenzen nur einseitig, beziehungsweise kombinieren zwei einseitige Betrachtungen wie zum Beispiel ELMO (Peters et al., 2018) und haben demnach keine tiefe bidirektionale Repräsentation der Sequenzen.

Beim Masked Language Modeling werden die bidirektionalen Repräsentationen trainiert indem zufällig 15% der Worttokens in einer Sequenz gegen ein **[MASK]** Token ersetzt werden. Das Trainingsziel ist, die maskierten Tokens korrekt vorherzusagen. Zum Verständnis der Zusammenhänge zwischen den Sätzen wird beim Next Sentence Prediction Pretrainingsschritt vorhergesagt, ob ein zu 50% zufällig gewählter Satz B auf Satz A folgt (Devlin et al., 2018).

## 3.4 Generative Pre-trained Transformer-2

GPT-2 (**G**enerative **P**re-trained **T**ransformer-2) ist ein von OpenAI (Radford et al., 2019) veröffentlichtes autoregressives Sprachmodell auf Basis von Transformerdecodern. Autoregressive Sprachmodelle generieren Textsequenzen, indem sie einzeln Tokens generieren und diese an die Eingabesequenz anhängen. Die neue verlängerte Textsequenz wird erneut als Eingabe zur Generation genutzt. Als generatives Modell erzielt es hervorragende Ergebnisse beim Generieren von kongruenten Textpassagen.

GPT-2 small besteht aus 12 sequentiellen Transformerdecodern. Um die autoregressiven Eigenschaften beizubehalten, verwendet GPT-2 Masked Self-Attention Layer. Durch die Masked Self-Attention Layer kann das GPT-2 Modell beim Training im Gegensatz zu BERT lediglich die bereits bekannten Tokens betrachten.

Zur Generation von Textpassagen kann GPT-2 unkontrolliert Samples aus einer leeren Eingabesequenz generieren. Kontrolliert werden kann die Generation durch das Festlegen einer spezifischen Starteingabe an die GPT-2 generierte Tokens anfügt. Während der

Generation selbst lässt sich die Generation allerdings nicht kontrollieren.

GPT-2 verwendet zur Tokenisierung das BytePairEncoding Verfahren (Sennrich et al., 2015). Das Pretraining von GPT-2 wurde auf einem Datensatz mit ca. 8 Millionen Webseiten durchgeführt, mit der Aufgabenstellung jeweils in Texten das nächste Token vorherzusagen.

# 4 Variational Autoencoder

Variational Autoencoder (Diederik P Kingma und Welling, 2014) gehören zu den probabilistischen generativen Modellarchitekturen. In den letzten Jahren haben generative Modelle, insbesondere auch Variational Autoencoder, beeindruckende Möglichkeiten aufgezeigt, hochrealistische Daten wie zum Beispiel Bilder, Texte oder Audios zu generieren. Generative Modelle versuchen die Merkmale und Verteilung eines Datensatzes zu verstehen und anschließend neuartige Datenbeispiele, die ähnlich zum Trainingsdatensatz sind zu generieren.

Autoencoder sind neuronale Netze die trainiert werden um Eingabedaten zu komprimieren und anschließend zu rekonstruieren. Sie bestehen aus einem Encoder $E : \mathbb{R}^n \to \mathbb{R}^m$ und einem Decoder$D : \mathbb{R}^m \to \mathbb{R}^n$, wobei der Encoder versucht, eine komprimierte Darstellung $c \in \mathbb{R}^m$ für die Eingabedaten $x \in \mathbb{R}^n$ im Latentspace zu finden. Die komprimierte Darstellung im Latentspace wird vom Decoder rekonstruiert mit dem Ziel, eine möglichst identische Rekonstruktion zur Eingabe $x \in \mathbb{R}^n$ zu erzeugen. Um stets eine komprimierte Darstellung im Latentraum zu erhalten und nicht eine Identitätsabbildung für den Encoder und Decoder wird die Dimension der Latentrepräsentation niederiger als die Eingabedimension gewählt $m < n$. Da Autoencoder nur einen diskreten Latentraum erzeugen existieren im Latentraum viele leere Stellen, wodurch Interpolation durch den Latentraum zur Rekonstruktion neuer Ausgaben kaum möglich ist. Dies ist darauf zurückzuführen, dass ein Autoencoder auf möglichst geringen Verlust beim Encodieren und Decodieren trainiert wird und somit keinen Wert auf die Organisation des Latentraums legt.

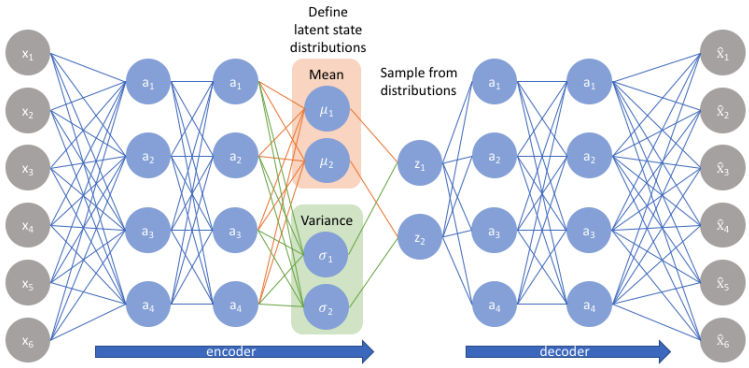



Abbildung 3: VAE Modellarchitektur (Jordan, 2018)

Im Gegensatz zu normalen Autoencodern verwenden Variational Autoencoder einen probabilistischen Ansatz, um Datenpunkte $x_i$ eines Datensatzes $X$, der durch die unbekannte Wahrscheinlichkeitsfunktion $P(\mathbf{X})$ beschrieben wird, im Latentraum $Z$ zu repräsentieren. Für den Datensatz $X$ mit Datenpunkten $x_i$ wird angenommen, dass ein generatives Modell mit den Parametern $\theta$ existiert, welches mittels Latentvektor $z_i$ diesen Datenpunkt $x_i$ generieren kann. Dieser generative Teil des Variational Autoencoders ist der Decoder des Netzes, welcher approximativ versucht die Daten als Verteilung $p_\theta(\mathbf{X})$ zu modellieren.

$$p_\theta(\mathbf{x}) = \int_{\mathbf{z}} p_\theta(\mathbf{x} \mid \mathbf{z}) p_\theta(\mathbf{z}) dz \tag{5}$$

In Gleichung 5 wird für $p_\theta(\mathbf{z})$ die A-priori-Wahrscheinlichkeit angenommen und für $p_\theta(\mathbf{x} \mid \mathbf{z})$ die Likelihood. Der Decoder zieht demnach eine Latentvariable $z$ aus der A-priori-Menge $p_\theta(\mathbf{z})$ und berechnet mit der Likelihood $p_\theta(\mathbf{x} \mid \mathbf{z})$ Datenpunkte $x$ (Bank et al., 2020).

Die inferierten Latentvariablen für die bekannten Datenpunkte $X$ lassen sich über die A-posteriori-Wahrscheinlichkeit $p_\theta(\mathbf{z} \mid \mathbf{x})$ aus Gleichung 6 herleiten.

$$p_\theta(\mathbf{z} \mid \mathbf{x}) = \frac{p_\theta(\mathbf{x} \mid \mathbf{z}) p_\theta(\mathbf{z})}{p_\theta(\mathbf{x})} \tag{6}$$

Der Encoder des Variational Autoencoder approximiert die A-posteriori-Wahrscheinlichkeit durch ein Inferenzmodell $q_\phi(\mathbf{z} \mid \mathbf{x})$, da sich $p_\theta(\mathbf{z} \mid \mathbf{x})$ nicht trivial berechnen lässt. Die Parameter $\phi$ des Encoders werden optimiert um mit $q_\phi(\mathbf{z} \mid \mathbf{x}) \approx p_\theta(\mathbf{z} \mid \mathbf{x})$ die wahre A-posteriori-Verteilung zu approximieren, indem die Kullback-Leibler-Divigenz zwischen den beiden Wahrscheinlichkeiten minimiert wird, wie in Abschnitt 4.1 beschrieben wird.

## 4.1 Evidence Lower Bound

Die Lossfunktion von Variational Autoencodern ist die **E**vidence **L**ower **Bo**und (ELBO). Bei Variational Autoencodern ist das Optimierungsziel die gleichzeitige Minimierung des Rekonstruktionsfehlers des generativen Modells $\theta$ zwischen Eingabe- und Ausgabedaten sowie die Approximation von $p_\theta(\mathbf{z} \mid \mathbf{x})$ mit $q_\phi(\mathbf{z} \mid \mathbf{x})$ durch Minimierung der Kullback-Leibler-Divergenz$D_{KL}(q_\phi(\mathbf{z} \mid \mathbf{x}) \parallel p_\theta(\mathbf{z} \mid \mathbf{x}))$. Für eine beliebige Wahl der Decoder Parameter $\phi$ gilt (Diederik P. Kingma und Welling, 2019):

$$\begin{aligned}
log(p_\theta(\mathbf{x})) &= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[log(p_\theta(\mathbf{x}))] && (7)\\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[log\left[\frac{p_\theta(\mathbf{x},\mathbf{z})}{p_\theta(\mathbf{z}|\mathbf{x})}\right]\right] && (8)\\
&= \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[log\left[\frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})}\right]\right] && (9)\\
&= \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[log\left[\frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right]\right]}_{=\mathcal{L}_{\theta,\phi}(\mathbf{x})\text{ (ELBO)}} + \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[log\left[\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})}\right]\right]}_{=D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x}))} && (10)
\end{aligned}$$

In Zeile 10 ist der erste Term die Evidence Lower Bound und der zweite Teil die Kullback-Leibler Divergenz. Die Kullback-Leibler Divergenz zwischen $q_\phi(\mathbf{z} \mid \mathbf{x})$ und $p_\theta(\mathbf{z} \mid \mathbf{x})$ ist nicht negativ und genau null, wenn $q_\phi(\mathbf{z} \mid \mathbf{x})$ der wahren A-posteriori-Verteilung entspricht.

$$D_{KL}(q_\phi(\mathbf{z} \mid \mathbf{x}) \parallel p_\theta(\mathbf{z} \mid \mathbf{x})) \geq 0 \tag{11}$$

Aus Gleichung 10 wird die Evidence Lower Bound Funktion aufgestellt:

$$\mathcal{L}_{\theta,\phi}(\mathbf{x}) = log(p_\theta(\mathbf{x})) - D_{KL}(q_\phi(\mathbf{z} \mid \mathbf{x}) \parallel p_\theta(\mathbf{z} \mid \mathbf{x})) \leq log(p_\theta(\mathbf{x})) \tag{12}$$

Aufgrund der nicht Negativität der Kullback-Leibler Divergenz in Gleichung 11 ist die Evidence Lower Bound eine untere Schranke der Log-Likelihood der Datenmenge. Bei

Maximierung der Evidence Lower Bound Funktion in Gleichung 12 wird die Evidenz $log(p_\theta(\mathbf{x}))$ maximiert, wodurch das generative Modell verbessert wird. Ebenfalls wird die Kullback-Leibler Divergenz minimiert, wodurch der Encoder die A-posteriori-Verteilung besser approximieren kann.

Somit wird als Lossfunktion $\mathbf{L}_{\theta,\phi}$ die negative Evidence Lower Bound Funktion gewählt, die es zu minimieren gilt:

$$\mathbf{L}_{\theta,\phi} = -\mathcal{L}_{\theta,\phi}(\mathbf{x}) = -log(p_\theta(\mathbf{x})) + D_{KL}(q_\phi(\mathbf{z} \mid \mathbf{x}) \parallel p_\theta(\mathbf{z} \mid \mathbf{x})) \tag{13}$$

$$\hat{\theta}, \hat{\phi} = \arg\min_{\theta,\phi} \mathbf{L}_{\theta,\phi} \tag{14}$$

Diese Lossfunktion erlaubt das simultane Optimieren der beiden Parameter $\theta$ und $\phi$.

### 4.1.1 Reparametisierung

Zum Training des Variational Autoencoder wird mittels Backpropagation der Fehler des Netzwerkes propagiert. Da das Samplen von $z \sim q_\phi(\mathbf{z} \mid \mathbf{x})$ nicht deterministisch ist, kann hier keine Backpropagation durchgeführt werden. Um dennoch Backpropagation verwenden zu können wird mittels Reparametisierung $z$ durch eine deterministische Funktion $z = f_\phi(x, \epsilon)$ dargestellt mit $\epsilon$ als externe unabhängige Hilfsvariable (Diederik P Kingma und Welling, 2014; Jordan, 2018). Bei einer multivariaten Gaussverteilung für $q_\phi(\mathbf{z} \mid \mathbf{x})$ wäre die Reparametisierung wie folgt, wobei $\times$ die elementweise Multiplikation denotiert:

$$\mathbf{z} \sim q_\phi(\mathbf{z} \mid \mathbf{x}) = \mathcal{N}(\mathbf{z}; \mu, \sigma\mathcal{I}) \tag{15}$$

$$\mathbf{z} = \mu + \sigma \times \epsilon \text{ , mit } \epsilon \sim \mathcal{N}(0, \mathcal{I}) \tag{16}$$

## 4.2 Cyclical Annealing Schedule

Trainieren von Variational Autoencodern als Sprachmodell im Bereich des Natural Language Processing ist aufgrund des KL-Vanishing Problems besonders schwierig. VAE Sprachmodelle sollen bei der Textgeneration den lokalen Kontext, allerdings auch globale Eigenschaften wie zum Beispiel Thema, Sprachstil oder Stimmung beachten. Trotzdem werden von VAE Modellen bei der Generation oft die globalen Eigenschaften vernachlässigt. Dieses Problem ist als KL-Vanishing bekannt, da die KL-Regularisierung des Loss-Terms beim Trainieren von autoregressiven Decodern innerhalb von VAE Sprachmodellen sehr klein wird. Somit sind die erlernten Features nahezu identisch zu der vorher angenommenen Normalverteilung und der Decoder nutzt keine Latentfeatures bei der Generation.

Abhilfe schafft die Verwendung eines $\beta$-Variational Autoencoder (Fu et al., 2019) mit zyklischem Erhöhen des $\beta$ Wertes. $\beta$-Variational Autoencoder sind Variational Autoencoder, die mit einen Lagrangemultiplikator $\beta$ die KL-Divergenz der Loss-Funktion gewichten, um besser separierte Latentfaktoren zu finden.

$$\mathbf{L}_\beta(\beta, \theta, \phi) = -\mathbb{E}_{z \sim q_\phi(\mathbf{z}|\mathbf{x})}[logp_\theta(\mathbf{x} \mid \mathbf{z})] - \beta D_{KL}(q_\phi(\mathbf{z} \mid \mathbf{x}) \parallel p_\theta(\mathbf{z})) \tag{17}$$

Falls $\beta = 0$ ist wird das Modell wie ein normaler Autoencoder trainiert, bei $\beta = 1$ wie ein normaler Varitional Autoencoder.

Beim zyklischen Erhöhen wird der $\beta$-Wert des VAE während des Trainings monoton von $\beta = 0$ auf zum Ende des Trainings $\beta = 1$ in kleinen Abständen erhöht. So kann beim Training des VAE bei $\beta < 1$ zunächst der Fokus darauf gelegt werden, mehr Information für die Rekonstruktion zu erlernen. Abschließend enthalten bei $\beta = 1$ die vorher trainierten $z$ Vektoren bereits mehr Informationen, die zu einer besseren Anpassung als eine zufällige Initialisierung führen.

## 4.3 Optimus

Optimus (**O**rganizing **S**entences via **P**re-trained **M**odeling of a **L**atent **S**pace) (Li et al., 2020) ist ein großes vortrainiertes Deep Latent Variable Modell für natürliche Sprache. Die Modelarchitektur von Optimus ist ein Variational Autoencoder, der als Encoder BERT und als Decoder GPT-2 verwendet wie in Abbildung 4 zu erkennen ist. Verwendet wird jeweils das vortrainierte BERT Base Modell $\phi_{BERT}$ und das vortrainierte GPT-2 Base Modell $\theta_{GPT-2}$ beide mit 12 Layern, 12 Attention Heads und einer Hiddensize von 768. Trainiert wird Optimus mit dem Ziel, Sätze in einem universellen Latentspace zu

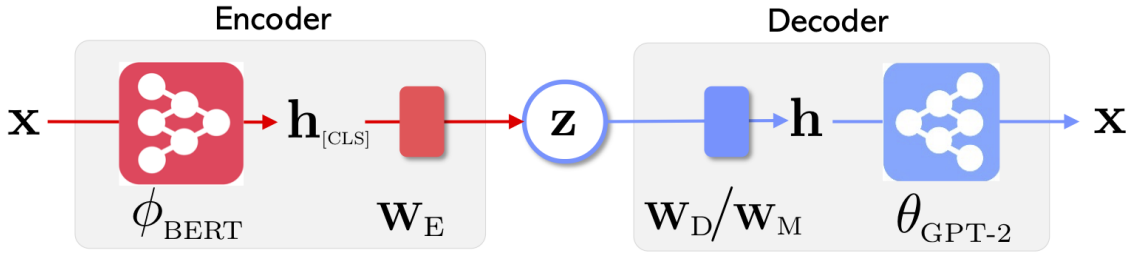
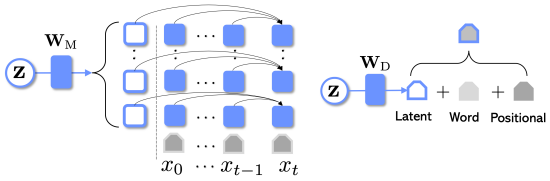



Abbildung 4: VAE Modellarchitektur von Optimus mit BERT als Encoder und GPT-2 als Decoder (Li et al., 2020)

organisieren und somit übergreifende semantische Muster für die einzelnen Sätze zu finden. Somit kann durch gezieltes Verändern des Latentvektors $z$ kontrolliert Text generiert werden.

BERT und GPT-2 über eine VAE Architektur miteinander zu verbinden hat die Herausforderung, die unterschiedlichen Tokenisierungsschemen der einzelnen Modelle zu verwenden und den Latentvektor bei der Textgeneration von GPT-2 zu injizieren. Die Eingabetokens von BERT verwenden das WordPiece Embeddings Verfahren (Wu et al., 2016) mit einer Vokabulargröße von 28.996. Die Ausgabe erfolgt über die Byte Pair Encoding Tokenisierung (Sennrich et al., 2015) von GPT-2 mit einer Vokabulargröße von 50.260. Innerhalb des Netzwerkes wird im Latentvektor ein Token durch eine Einbettung $h_{Emb}$, die das Token, die Position und das Segment Embedding wiedergibt, repräsentiert. Um beim Training den Loss des Rekonstruktionstask zu berechnen, werden die Sätze mit beiden Tokenisierungen tokenisiert.

Als Latentvektor $z \in \mathbb{R}^P$ wird die gepoolte Ausgabe des letzten Hiddenlayers $h_{[CLS]} \in \mathbb{R}^H$ von BERT mit einer Gewichtsmatrix multipliziert $W_E \in \mathbb{R}^{P \times H}$ gewählt. Somit kann ein Latentvektor wie folgt bestimmt werden $z = W_E h_{[CLS]}$.




(a) Memory (b) Embedding

Abbildung 5: Methoden, um den Latentvektor in GPT-2 zu injizieren (Li et al., 2020)

Um mit GPT-2 kontrolliert unter Verwendung des Latentvektors Text zu Generieren, kann der Latentvektor entweder als Memoryvektor in die Past-Gewichtsmatrix injeziert oder auf die vorherigen Embeddings addiert werden (Li et al., 2020).

Beim Embedding wird $z$ direkt auf den Embedding Layer addiert. Somit ergibt sich der neue Embedding Layer durch $h'_{Emb} = h_{Emb} + W_D z$ mit der Gewichtsmatrix $W_D \in \mathbb{R}^{H \times P}$. Der Decoder kann hier die zusätzlichen Informationen des Embeddings beim Input und Output Layer verwenden.

Bei der Injezierung von $z$ als Memoryvektor $h_{Mem} \in \mathbb{R}^{L \times H}$ wird der Latentvektor in den Past-Key-Vektor von GPT-2 injeziert. Der Past-Key-Vektor beschleunigt normalerweise den Decodiervorgang von GPT-2, da bei einem Decodierungsdurchlauf zu den vorherigen Tokens bereits berechnete Key- und Valuevektoren der Attentionlayern gespeichert und wiederverwendet werden. Diese Key-, Valuevektoren des Attentionlayers werden durch $h_{Mem} = W_M z$ mit $W_M \in \mathbb{R}^{LH \times P}$ ersetzt. GPT-2 kann so beim Decodieren auf den injezierten Latentvektor bei jeder Attentionberechnung zugreifen.

Die Parameter $\phi_{BERT}, \theta_{GPT-2}, W_E, W_M, W_D$ des VAEs werden mittels Cyclical Annealing Schedule (Fu et al., 2019) trainiert, um das KL Vanishing Problem beim Trainieren eines VAEs zu verhindern. Die $\beta$ Variable, die wie in 4.2 erklärt den KL-Regularisierer Anteil steuert, wird für einen Trainingsdurchlauf für die erste halbe Menge der Trainingsdaten auf $\beta = 0$ gesetzt. Somit wird zu Beginn lediglich ein Autoencoder trainiert. Anschließend wird $\beta$ für das nächste Viertel der Trainingsmenge schrittweise von 0 auf 1 erhöht und für das letzte Viertel der Trainingsmenge auf 1 belassen, um das VAE Modell zu trainieren.

Insgesamt zeigt Optimus gute Ergebnisse in den Bereichen des Language Modeling, der kontrollierten Text Generation und des Language Understanding.

## 4.4 BiMeanVAE

BiMeanVAE ist ein von (Iso et al., 2021) vorgestelltes Modell, bestehend aus einem bidirektionalem LSTM Encoder und einem LSTM Decoder. Die Hiddensize der einzelnen LSTMs beträgt 512. Nach dem BiLSTM Encoder wird ein mean pooling verwendet, um einen Latentvektor $z$ zu erhalten. BiMeanVAE verwendet die Standard Variational Autoencoder Ziele, dass Minimieren des Rekonstruktionsfehlers mit KL-Regularisierung verwendet. Die verwendete Prior Verteilung $p_\theta(z)$ ist die Gausssche Normalverteilung $\mathcal{N}(0, \mathcal{I})$. Beim Trainig wird ebenfalls das in Abschnitt 4.2 beschriebene Cyclical Annealing Verfahren verwendet, um graduell vom Autoencoder zum Variational Autoencoder Trainingsziel zu wechseln.

## 4.5 Latent Space Operationen

VAEs erlernen bidirektionale Abbildungen zwischen dem Datenraum und ihrem Latentspace. Im Latentspace werden Eingabesequenzen von Optimus und BiMeanVAE jeweils durch einen Latentvektor dargestellt. Die zur Generation verwendeten Latentvektoren können durch arithmetische Operationen verändert werden, wodurch sich die gesampelte Ausgabe des VAEs präzise verändern und steuern lässt.

Durch die KL-Regularisierung beim Training von VAEs entsteht ein kontinuierlicher Latentspace, wodurch semantisch ähnliche Inhalte im Latentspace nah beieinander organisiert werden. Somit kann zwischen mehreren Datenobjekten im Latentspace interpoliert werden.

Insbesondere für das kontrollierte Generieren von Bewertungen sind diese Latent Space Operationen, die VAEs ermöglichen elementar. So kann zum Beispiel zwischen mehreren Bewertungen interpoliert werden, um eine durchschnittliche Meinung zu generieren.

Sei zum Beispiel $\mathcal{Z}$ die Menge der encodierten Bewertungen zu einer Produktgruppe. So lässt sich aus einer Untergruppe der Vektoren $\mathcal{Z}' \subseteq \mathcal{Z}$ durch geschicktes Kombinieren eine gute Durchschnittsrepräsentation im Latentvektorraum finden. Das genaue Vorgehen zum Finden einer solchen Untergruppe wird in Abschnitt 6.1 beschrieben

# 5 Datensätze

Zum Training und zur Evaluation von neuronalen Netzen sind große Datensätze erforderlich. Für die Aufgabenstellung, Durchschnittsrezensionen aus Textbewertungen zu erzeugen, werden Datensätze mit mehreren Bewertungen und Zusammenfassungen zu einem Produkt benötigt.

Es bietet sich an, Bewertungen von großen Webportalen wie Amazon oder Yelp zu verwenden. Diese Portale haben zu den unterschiedlichsten Produkten und Restaurants zahlreiche Bewertungen.

Es existieren bereits bestehende Amazon und Yelp Datensätze mit menschlich erstellten Gold-Standard Zusammenfassungen. Die Variational Autoencoder Modelle lassen sich somit mit den Review Daten mit dem Trainingsziel der Rekonstruktion trainieren, um einen aussagekräftigen Latentraum mit Bewertungen und deren spezifischen Eigenschaften zu erlernen. Anschließend können die trainierten Modelle auf den respektiven Testdatensätzen evaluiert werden, da jeweils Gold-Standard Zusamenfassungen vorliegen, mit denen sich mittels in Abschnitt 8 erklärten Metriken die generierten Rezensionen bewerten lassen.

Der Umfang der zum Training und Testen verwendeten Datensätze ist in Tabelle 1 angegeben.

| | **Amazon** | | | **Yelp** | | |
|---|---|---|---|---|---|---|
| | Train | Dev | Test | Train | Dev | Test |
| # Produkte / Restaurants | 244.652 | 28 | 32 | 75.988 | 100 | 100 |
| # Rezensionen | 13.053.202 | 224 | 256 | 4.658.968 | 800 | 800 |

Tabelle 1: Enthaltene Rezensionen der einzelnen Datensätze (Iso et al., 2021)

## 5.1 Amazon Datensatz

Der Amazon Review Datensatz (Bražinskas et al., 2020) umfasst zahlreiche Rezensionen zu den unterschiedlichsten Produkten die auf dem Amazon Marktplatz gelistet sind. Die Rezensionen wurden von Amazon Nutzern erstellt und geben eine unabhängige Meinung über das erworbene Produkt ab. Bei Betrachtung der einzelnen Bewertungen fällt auf, dass die meisten Bewertungen von Produkten eher objektiv formuliert sind. Die Bewertungen gehen oft auf unterschiedliche Aspekte der Produkte ein und erläutern positive, sowie negative Erfahrungen die auf einzelne Produkteigenschaften zurückzuführen sind.

Die für das Training relevanten Bewertungen wurden aus den Kategorien Kleidung, Schuhe, Schmuck, Elektronikartikeln, Gesundheit- und Pflegeprodukte, Einrichtung und Küchenartikeln gewählt. Gefiltert wurden die Bewertungen indem Produkte mit einer maximalen Länge von 128 Tokens gewählt wurden und nur Produkte mit über 10 existierenden Bewertungen ausgewählt wurden. Die entsprechenden Bewertungen sind bereits in Trainings- und Validierungsdaten gesplittet. Die Validierungsdaten des Amazon

**Produkt:** NuGo Protein Bar, Vanilla Yogurt

**Review:** Apart from the obvious nutritional value, the taste was surprisingly better than I anticipated. Normally, if I find soy-based nutritional bars tend to have a chalky or tree-bark-like texture and taste. I dare say, the texture reminds me of a healthy-not-too-sweet 'Rice Krispy Treat'.

Abbildung 6: Beispiel Bewertung zu einem Produkt des Amazon Datensatzes

Datensatz sind manuell erstellte Zusammenfassungen, die für den Dev / Test Split im Verhältnis von 28 / 32 vorliegen. Somit ergibt sich insgesamt ein Trainingsdatensatz mit 244.652 Produkten und den zugehörigen 13.053.202 Bewertungen.

## 5.2 Yelp Datensatz

Der Yelp Review Datensatz (Chu und Liu, 2019) enthält Rezensionen zu unterschiedlichen Gastronomiebetrieben und Unternehmen der Bewertungsplattform Yelp. Die von den Yelpnutzern erstellten Bewertungen unterschieden sich von den Amazon Bewertungen durch mehr Subjektivität in den Bewertungen. Die Bewertungen enhtalten wesentlich mehr persönliche Details und Erfahrungen der Nutzer. Das Variational Autoencoder Modell hat hier die Aufgabe, wichtige Informationen zu extrahieren und unwichtiges Rauschen in den Daten herauszufiltern, um eine gute Durchschnittsrepräsentation zu bestimmen. Insbesondere in den Yelp Reviews befinden sich meistens unnötige Zusatzinformationen, die nicht zur Bewertung des Restaurants an sich beitragen, wie zum Beispiel das Erwähnen von privat ausgetragenen Geburtstagsfeiern.

**Restaurant Id:** 87YsVbCN_kfzheY79Fzjkg

**Review:** Great experience! Went cake tasting for our wedding and Jessica was a huge help!! She brought out the tastings for us after giving us time to look through their amazing book. She explained every cake and filling and had great recommendations for mixing the fillings. The champagne cake with cream cheese filling was amazing!! Chose our cake and design within an hour. Great service and knowledge. Thank you Jessica!

Abbildung 7: Beispiel Bewertung zu einem Restaurant des Yelp Datensatzes

Der Yelp Review Datensatz wurde ebenfalls nach einer maximalen Länge von 128 Tokens mit mindestens 10 existierenden Bewertungen je Unternehmen gefiltert und besteht somit aus 13.053.202 Bewertungen zu 244.652 Unternehmen, die bereits in Trainings- und Validierungsdaten gesplittet sind. Zusätzlich enthält der Datensatz 200 menschlich durch Amazon Mechanical Turk (AMT) Arbeiter erstellte Zusammenfassungen, die zu einem Dev / Test Verhältnis von 100 / 100 gesplittet werden.

# 6 Multi-Document Summarization

In den letzten Jahren erzielten unterschiedliche Verfahren große Fortschritte beim automatisierten Zusammenfassen von mehreren Dokumenten zu einem repräsentativen Dokument. Im Gegensatz zum normalen Zusammenfassen von einzelnen Dokumenten enthalten beim Multi-Document Zusammenfassen mehrere Dokumente über ein Thema viele redundante Informationen, die so gefiltert werden müssen, dass redundante Informationen nur einmal im Ausgabedokument dargestellt werden. Ebenfalls ist das Verarbeiten von widersprüchlichen Informationen eine große Herausforderung bei der Zusammenfassung von mehreren Dokumenten. Das Ziel von Multi-Document Summarization ist das zusammengefasste Repräsentieren der wichtigen Aspekte aller Dokumente in einem einzelnen Dokument, um einen guten allgemeinen Überblick zu schaffen.

Im weiteren Verlauf werden Multi-Document Summarization Systeme zur Zusammenfassung von Bewertungen verwendet. Sei $R = \{x_i\}$ ein Datensatz von Bewertungen mit einzelnen Bewertungen $x = (x_1, ..., x_{\|x\|})$, die aus einer Sequenz aus Wörtern bestehen. Ziel ist es für ein gegebenes Produkt $p$ und die entsprechenden Bewertungen $R_p \subseteq R$ eine Zusammenfassung $s_p$ zu generieren, die alle relevanten Informationen faktisch korrekt representiert.

Insbesondere im Bereich der Multi-Review Summarization existieren viele unterschiedliche Ansätze. Zum einen existieren extraktive Ansätze, wie zum Beispiel LexRank (Erkan und Radev, 2011), ein unüberwachter Algorithmus, der repräsentative Sätze für eine Bewertung basierend auf ihrer Zentralität in einem TF-IDF gewichteten Graphen selektiert. Zum anderen existieren abstraktive Ansätze wie zum Beispiel MeanSum (Chu und Liu, 2019), CopyCat (Brazinskas et al., 2019) oder COOP (Iso et al., 2021), die generative Ansätze verwenden, um neuartige Sätze in den Bewertungen zu erzeugen.

Die abstraktiven Modelle basieren auf Autoencoder Architekturen. Es wird ein Encoder $E_\phi : X \to Z$ verwendet, um Textrepräsentationen in einen Latentvektor im Latentraum $Z$ umzuwandeln. Aus den Latentvektoren $z$ kann anschließend unter Verwendung von einem Decoder $D_\theta : Z \to \hat{X}$ eine Textrepräsentation generiert werden.

MeanSum ist ein Autoencoder Modell (Chu und Liu, 2019) mit LSTM Encoder und Decoder und errechnet zur Zusammenfassung von Bewertungen den Durchschnitt von den einzelnen Latentvektoren der Bewertungen $z_p = \bar{z}$, mit $z = \{z_{p_1}, z_{p_2}, ...z_{p_k}\}$. Von diesem Durchschnittslatentvektor $z_p$ wird anschließend eine Bewertung generiert.

CopyCat basiert auf einem Variational Autoencoder Modell (Brazinskas et al., 2019), welches GRU Encoder und Decoder verwendet. Für jede Gruppe von Bewertungen berechnet CopyCat einen Latentvektor $c$, der die gesamte Semantik der Gruppe beschreibt. Weiterhin wird jede einzelne Bewertung einer Gruppe mit einem einzelnen Latentvektor $z$ beschrieben. Bei der Generation von Durchschnittsbewertungen ermöglicht es CopyCat dem Decoder, die einzelnen Latentvektoren für die Bewertungen $z_i$, den allgemeinen Gruppenvektor $c$ und die Bewertungen $r_i$ an sich zu betrachten. Durch den Zugriff auf die anderen Bewertungen $r_i$ kann der Decoder spezifische Worte von diesen „kopieren“ und somit übernehmen.

## 6.1 Convex Aggregation for Opinion Summarization

Convex Aggregation for Opinion Summarization kurz COOP (Iso et al., 2021) ist eine Methode, die unterschiedliche Kombinationenen von einzelnen Latentvektoren zum Berechnen einer Durchschnittsbewertung untersucht. Bei der Verwendung von Variational Autoencodern werden die einzelnen Eingabebewertungen durch einen Latentvektor repräsentiert. Häufig wird im nächsten Schritt zur Berechnung einer Durchschnittsbewertung der normale Durchschnitt aller verwendeten Latentvektoren bestimmt und von diesem eine neue Durchschnittsbewertung generiert.

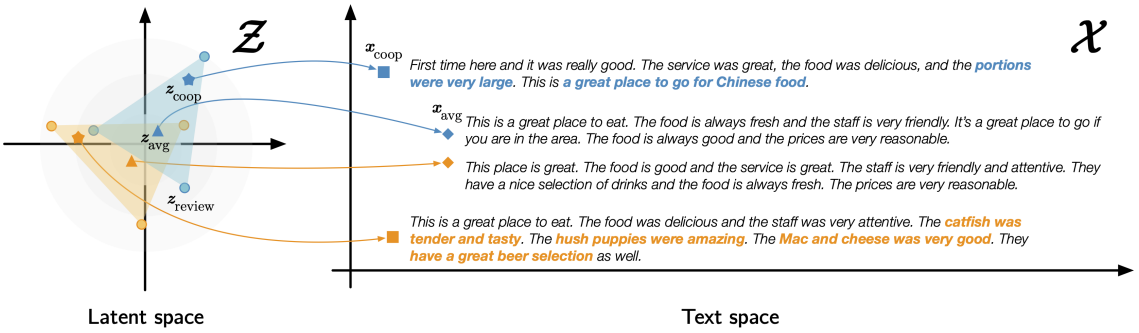



Abbildung 8: Latentraum $Z$ mit den entsprechenden generierten Bewertungen $X$ (Iso et al., 2021)

Wie in Abbildung 8 zu erkennen ist ähneln sich Bewertungen, die durch Bestimmung des Durchschnitts der Latentvektoren generiert worden sind sehr. Auffällig ist die Korrelation zwischen der Ausdrucksstärke und dem Informationsgehalt eines Latentvektors und seiner $L_2$-Norm $\|z\|$. Latentvektoren, die durch den Durschnitt bestimmt werden, haben eine geringere $L_2$-Norm und somit in den erzeugten Bewertungen einen geringeren Informationsgehalt (Iso et al., 2021).

Um ausdrucksstarke Latentvektoren für die Generation der Bewertungen zu erhalten, formuliert COOP (Iso et al., 2021) die Bestimmung des optimalen Latentvektors als Optimierungsproblem, um die beste Kombination von einzelnen Latentvektoren zu finden.

$$\max_{z} \quad Overlap(R_p, D_\theta(z)) \tag{18}$$

unter der Nebenbedingung:

$$z = \sum_{i=1}^{|R_p|} w_i z_i \tag{19}$$

$$\sum_{i=1}^{|R_p|} w_i = 1, \quad \forall w_i \in \mathbb{R}^+ \tag{20}$$

Maximiert wird der *Input-Output-Word Overlap* zwischen den Eingabebewertungen $R_p$ und der generierten Bewertung $D_\theta(z)$. Die *Overlap*-Metrik in Gleichung 21 basiert auf dem ROUGE-1 F1 Score und ermöglicht es, Bewertungen zu generieren, die konsistent zu den Eingabebewertungen sind.

$$Overlap(X, Y) = \text{ROUGE-1}_{F_1}(X, Y) \tag{21}$$

Die Suche der einzelnen Kombinationen wird auf die Potenzmenge der Eingabebewertungen $R_p$ beschränkt. Der Zusammenfassunslatentvektor $z_p$ wird anschließend aus dem Durchschnitt der ausgewählten Eingabebewertungslatentvektoren berechnet.

$$z_p = \frac{1}{|R_p^{'}|} \sum_{i=1}^{R_p^{'}} z_i, \text{ mit } R_p^{'} \in 2^{R_p} \setminus \{\emptyset\} \tag{22}$$

Die COOP Methode zur Kombination der einzelnen Latentvektoren, um ausdrucksstarke Bewertungen zu erhalten, erreicht State-of-the-Art Ergebnisse im Vergleich zu den anderen vorgestellten Methoden, die lediglich den normalen Durchschnittslatentvektor verwenden. Die erzeugten Bewertungen sind repräsentativ für eine Gruppe von Bewertungen und erhalten ein hohes Maß an Informationen. Trotzdem stellt sich die Frage, ob und inwiefern sich diese Ergebnisse noch weiter optimieren lassen. Im weiteren Verlauf wurde ein Verfahren entwickelt, welches entsprechende Latentvektoren weiterhin adaptiert, um ein noch höheres *Input-Output-Overlapping* zu erhalten und somit präzisere Informationen in den Durchschnittsbewertungen zu generieren.

Die beiden Modelle Optimus und BIMEANVAE sind bereits von den Autoren (Iso et al., 2021) vortrainiert und werden im weiteren Verlauf verwendet. Als Optimierer wurde Adam verwendet mit einer Learningrate von $10^{-5}$ für Optimus und $10^{-3}$ für BIMEANVAE. Die Cyclical Annealing Learning Strategie wurde wie in Kapitel 4.2 beschrieben angewandt.

# 7 Kontrollierbare Textgeneration von Sprachmodellen

Die in Abschnitt 6.1 vorgestellte COOP Methode zur Suche der optimalen Kombination von Latentvektoren zur Maximierung des *Input-Output-Overlaps* erzielt beeindruckende Resultate. COOP verwendet die VAE Modelle Optimus und BIMEANVAE, welche zur Generation den kombinierten Latentvektor $z$ mittels Decoder $p_\theta(x|z)$ zu einem Text $\hat{x}$ rekonstruieren. Optimus verwendet zur Decodierung ein GPT-2 Modell. BIMEANVAE decodiert den Latentvektor mit einem LSTM Decoder.

Obwohl durch die Kombination mehrerer Latentvektoren bereits gute Ergebnisse erzielt werden, stellt sich die Frage, ob diese kombinierten Latentvektoren durch weitere Optimierungen bessere Ergebnisse erzielen.

Unkontrollierte Sprachmodelle modellieren Texte über die Wahrscheinlichkeit $p(X)$ für eine Sequenz $X = (x_0, ..., x_{\|x\|})$. In Kapitel 3 wurde die Funktionsweise von Transformer-Sprachmodellen und die autoregressive Generation von Textsequenzen erklärt. Bei der Generation werden die vorherigen Key-Value Paare der Attention-Layer in einer Vergangenheitsmatrix $H_t = [(K_t^{(1)}, V_t^{(1)}), \ldots, (K_t^{(n)}, V_t^{(n)})]$ gespeichert, wobei $K$ und $V$ die einzelnen Key-, Value-Vektoren im Layer $n$ zum Zeitpunkt $t$ repräsentieren. Diese Vergangenheitsmatrizen werden verwendet, um bei der Generation auf bereits vorher berechnete Key-, Value-Werte zurückgreifen zu können und somit effizienter Text zu generieren.

Uber hat mit der Einführung von Plug and Play Language Models (Dathathri et al., 2019) es ermöglicht, die Textgeneration von großen Sprachmodellen wie zum Beispiel GPT-2 kontrolliert zu beeinflussen. Kontrollierbare Generation von Texten mittels Sprachmodellen entspricht dem Modellieren von $p(x|a)$, wobei hier $a$ für ein kontrollierbares Attribut in Bezug auf den generierten Text $x$ steht. Mit dem Satz von Bayes lässt sich das kontrollierbare Sprachmodell zu $p(x|a) \propto p(a|x)p(x)$ umformulieren (Dathathri et al., 2019). Das Attribut Modell $p(a|x)$ bewertet einen Satz $x$ auf den Besitz eines Attributs $a$ mit einer Wahrscheinlichkeit.

Zur kontrollierbaren Generation werden bei PPLM-Modellen Gradienten für die generierten Sequenzen über die Log-Likelihood des normalen Sprachmodells $log(p(x))$ und der Log-Likelihood des Attribut-Modells $log(p(a|x))$ in Bezug auf die Vergangenheitsmatrix errechnet. Durch Veränderung der Vergangenheitsmatrix $\tilde{H}_t = (H_t + \Delta H_t)$ wird die Wahrscheinlichkeit das nächste Token mit den gewünschten Attributen zu erhalten erhöht. Hierbei wird $\Delta H_t$ schrittweise durch den Gradienten des Attribut-Modells bestimmt und mit einer Null-Matrix initialisiert. Um den Gradienten des Attribut-Modells zu bestimmen, wird dieses zu $p(a|H_t + \Delta H_t)$ umformuliert mit folgendem Iterationsschritt 23 zur Berechnung:

$$\Delta H_t \leftarrow \Delta H_t + \alpha \frac{\nabla_{\Delta H_t} \log p(a|H_t + \Delta H_t)}{\|\nabla_{\Delta H_t} \log p(a|H_t + \Delta H_t)\|^\gamma} \tag{23}$$

Der Iterationsschritt 23 kann mehrmals hintereinander ausgeführt werden.

Die so durch die modifizierte Vergangenheitsmatrix $\tilde{H}_t$ bestimmte Ausgangsverteilung $\tilde{p}_{t+1}$ wird mit der nicht modifizierten Ausgangsverteilung $p_{t+1}$ kombiniert, um den generierten Text ebenfalls durch die Sprachmodell-Verteilung zu beeinflussen. Somit lässt sich

das nächste Token wie in Gleichung 24 beschrieben von den kombinierten Verteilungen $p_{t+1}$ und $\tilde{p}_{t+1}$ samplen:

$$\hat{x}_{t+1} \sim \frac{1}{\beta}(\tilde{p}_{t+1}^{\gamma} \cdot p_{t+1}^{1-\gamma}) \tag{24}$$

Durch Veränderung von $\gamma$ bei der Kombinierung lässt sich der Einfluss des unmodifizierten Sprachmodells auf die Ausgabe festlegen. Hier konvergiert bei $\gamma \to 1$ die Ausgabe zur Verteilung des modifizierten Sprachmodells und $\gamma \to 0$ gegen die Ausgabe des unmodifizierten Sprachmodells.

## 7.1 Bag of Words Attribut-Modell

Als Attribut Modell wird ein Bag of Words Modell verwendet, welches einen Loss über die Summe der Wahrscheinlichkeiten der einzelnen vorhergesagten Wörter bildet. Sei $\{w_0, \ldots, w_n\}$ eine Gruppe von Tokens, die ein bestimmtes Thema repräsentieren und $p_{t+1}$ die Ausgabeverteilung über die Tokens des Sprachmodells. Dann wird die Log-Likelihood des Attributmodells durch Gleichung 25 beschrieben:

$$\log p(a|x) = \log \left(\sum_{i=0}^{n} p_{t+1}[w_i]\right) \tag{25}$$

Um den *Input-Output-Overlap* zwischen den Eingabebewertungen und den generierten Bewertungen zu maximieren, kann das Bag of Words Modell auf unterschiedliche Weise erzeugt werden. Eine Methode ist es, die $k$ am häufigsten vorkommenden Wörter über alle Eingabebewertungen eines Produktes zu wählen. Vor dem Auswählen der häufigsten vorkommenden Wörter werden Stoppwörter entfernt.

Des Weiteren können die ausgewählten Bag of Words Tokens gewichtet werden, indem die $k$ häufigsten Wörter nach ihrer Anzahl der Vorkommnisse über eine Softmax-Funktion in eine Wahrscheinlichkeitsverteilung transformiert werden. Hierdurch erhalten besonders häufig vorkommende Wörter ein höheres Gewicht als weniger häufig vorkommende Wörter.

Die Bag of Words Menge kann auch durch anderweitige Keyword-Extraktionsmethoden wie zum Beispiel durch YAKE (Campos et al., 2020) generiert werden.

In Tabelle 2 wurden auf dem Amazon Dev-Datensatz die unterschiedlichen Verfahren, um Bag of Words Mengen zu erzeugen, evaluiert. Es zeigt sich, dass bei dem Bag of Words Attributmodell eine Wortanzahl von 150 optimal ist, um gute Ergebnisse zu erzielen. Eine höhere Wörteranzahl ergibt keine weitere Leistungssteigerung. Dies lässt sich darauf zurückführen, dass gar nicht mehr Wörter ausgewählt werden können und diese bereits den gesamten Text umfassen würden.

Des Weiteren fällt auf, dass entgegen der Erwartung eine leichte Gewichtung mittels Softmax Funktion nach der Anzahl der Vorkommnisse entsprechender Wörter keine Leistungssteigerung bietet. Keyword Extraction über das YAKE-Verfahren zeigt ebenfalls keine Leistungssteigerung.

| Modell | Amazon | | |
|---|---|---|---|
| **BIMEANVAE** | **R1** | **R2** | **RL** |
| Baseline | 39.50 | 8.62 | 22.79 |
| k = 50 | 40.72 | **9.06** | **23.40** |
| k = 150 | **40.75** | 9.03 | **23.40** |
| k = 500 | **40.75** | 9.03 | **23.40** |
| k = 50 + Softmax | 40.72 | **9.06** | **23.40** |
| k = 150 + Softmax | **40.75** | 9.03 | **23.40** |
| k = 500 + Softmax | **40.75** | 9.03 | **23.40** |
| YAKE | 40.72 | 9.00 | **23.40** |

Tabelle 2: Ergebnisse für die unterschiedlichen Attributionsmodelle mit BIMEANVAE auf dem Amazon Dev-Datensatz

Demnach wird im folgenden für die weitere Evaluierung auf dem Testdatensatz das Bag of Words Attributmodell mit der Wörterauswahl aus den $k = 150$ am häufigst vorkommenden Wörtern initialisiert.

## 7.2 Verbessern der Textgeneration von Optimus

Den Variational Autoencoder Optimus mit einem Attributions-Modell zu kombinieren ist aufgrund der Injektion des Latentvektors nicht ohne weiteres möglich. Optimus kann bereits unter Einbezug des Latentvektors Texte kontrolliert generieren $p(x|z)$. Die Berücksichtigung eines Attribut-Modells bei der Generierung des Textes entspricht $p(x|a,z) \propto p(a|x,z)p(x|z)$. Da der Latentvektor $z$ in die Vergangenheitsmatrix $H_t$ injeziert wird, wird der Latentvektor direkt optimiert und $\Delta z$ ergibt sich durch die folgende Iteration 26:

$$\Delta z \leftarrow \Delta z + \alpha \frac{\nabla_{\Delta z} \log p(a|z+\Delta z, z)}{\|\nabla_{\Delta z} \log p(a|z+\Delta z, z)\|^{\gamma}} \tag{26}$$

In Abschnitt 7.6 wird die vorgeschlagene Optimierung der Latentvektoren anhand der Test-Datensätze evaluiert und nach geeigneten Parametern untersucht.

## 7.3 Verbessern der Textgeneration von BIMEANVAE

BIMEANVAE ist ein Variational Autoencoder, bestehend aus einem bidirektionalem LSTM Encoder, gepaart mit einem LSTM Decoder. Der LSTM Decoder erhält als Eingabe den Latentvektor $z$. Der Hiddenstate $h_t$ und Cellstate $c_t$ wird aus dem Embedding des Latentvektor $z$ initialisiert und anschließend autoregressiv über die LSTM Architektur aktualisiert. Um diesen LSTM Decoder mit einem Attributmodell zu optimieren, bieten sich drei unterschiedliche Ansätze:

1. Optimieren über den Latentvektor $z$
2. Optimieren des vorherigen Hiddenstates $h_t$ vor der nächsten Berechnung

3. Optimieren des vorherigen Cellstates $c_t$ vor der nächsten Berechnung

Es ist möglich, zu allen drei aufgezählten Optionen einen Gradienten zur Veränderung der entsprechenden Variabel zu bestimmen.

Nachfolgend wurden alle drei Optionen auf dem Amazon-Testdatensatz evaluiert, um die zu optimierende Variabel mit der bestmöglichen Performance zu finden. Die gewählte Schrittgröße ist $\alpha = 0.1$. Bei der Selektion der Ausgabebewertung wurde die Bewertung ausgewählt, die die höchste ROUGE-1 Übereinstimmung mit den Zusammenfassungsbewertungen hat. Somit wird stets die bestmögliche generierte Bewertung ausgewählt.

| Modell | Amazon | | |
|---|---|---|---|
| **BIMEANVAE** | **R1** | **R2** | **RL** |
| Baseline | 39.50 | <u>8.62</u> | 22.79 |
| optimze $z$ | <u>40.04</u> | 8.35 | **23.64** |
| optimze $h_t$ | 39.96 | 8.34 | 22.94 |
| optimze $c_t$ | **40.81** | **8.91** | <u>23.49</u> |

Tabelle 3: Ergebnisse für die Optimierung der unterschiedlichen Variablen $z, h_t, c_t$ über ein Attributionsmodell

In Tabelle 3 sind die ROUGE-Scores, welche in Abschnitt 8 genauer erklärt werden, für die separiert optimierten Variablen $z, h_t, c_t$ aufgelistet. Beim Evaluieren der unterschiedlichen Optimierungsversuche zeigt sich, dass jede Optimierung bessere Ergebnisse als die unoptimierte Baseline auf den Gold-Summaries erzielt. Die größte Leistungssteigerung erzielt eine Optimierung der Variabel $c_t$. Somit beschreibt folgende Gleichung den Optimierungsschritt $\Delta c_t$ bei BIMEANVAE.

$$\Delta c_t \leftarrow \Delta c_t + \alpha \frac{\nabla_{\Delta c_t} \log p(a|c_t + \Delta c_t, c_t)}{\|\nabla_{\Delta c_t} \log p(a|c_t + \Delta c_t, c_t)\|^\gamma}$$

## 7.4 Latentvektoroptimisierung mit Beam Search

Beam Search ist ein Suchalgorithmus zum Auffinden eines Ausgabesatzes mit der höchsten Wahrscheinlichkeit. Im Gegensatz zur normalen Greedy Suche bei der nachfolgend das bestmögliche Token ausgewählt wird, wird beim Beam Search Algorithmus eine Gruppe von N bestmöglichen Tokens für eine Position ausgewählt. Somit wird ein Suchbaum erstellt, der eine maximale Breite von N hat und die Äste mit den N höchsten Wahrscheinlichkeiten weiterführt. Demnach ist Beam Search nicht optimal, da die bestmögliche Lösung möglicherweise nicht fortgeführt wird. Trotzdem lassen sich mit Beam Search erfolgreich gute Ausgabesätze finden, die einer hohen Wahrscheinlichkeit entsprechen.

## 7.5 Moverscore Ranking

Da dem Modell zur Bestimmung des *Input-Output-Overlaps* mittels ROUGE-1 nur die Eingabebewertungen zur Verfügung stehen, erhalten teilweise generierte Rezensionen mit gleicher Semantik aber anderen Formulierungen schlechte *Input-Output-Overlap*-Scores. Der in Abschnitt 8.2 vorgestellte Moverscore ermöglicht einen semantischen Vergleich von Textsequenzen. So können insbesondere Rezensionen mit unterschiedlich formulierten Meinungen, die die gleiche Aussage treffen, ein hoher Ähnlichkeitswert zugewiesen werden.

Zwischen den Eingabebewertungen und den Gold-Summaries besteht eine hohe semantische Ähnlichkeit, weshalb der Moverscore miteinbezogen wird, um ein besseres Ranking der Ausgabewertungen zu erzeugen. Des Weiteren wird die *Input-Output-Overlap* Funktion dahingehend abgeändert, dass sie ebenfalls die ROUGE-2 und ROUGE-L Scores miteinbezieht. Somit ergibt sich eine neue Rankingfunktion 28 zwischen den generierten Ausgabebewertungen $\hat{x}_i$ und den Eingabereviews $R$:

$$\textit{Input-Output-Overlap}(\hat{x}_i, R) = x \cdot \text{R1}(\hat{x}_i, R) + y \cdot \text{R2}(\hat{x}_i, R) + z \cdot \text{RL}(\hat{x}_i, R) \tag{27}$$

$$\text{SCORE}(\hat{x}_i) = \textit{Input-Output-Overlap}(\hat{x}_i, R) + v \cdot \textit{Moverscore}(\hat{x}_i, R) \tag{28}$$

Die Variablen $x, y, z, v$ geben die Gewichtung der einzelnen Metriken an. Die Hyperparameter für die Gewichtung werden im folgenden Abschnitt auf dem Dev-Datensatz ermittelt.

Mit dieser neuen Rankingfunktion wird bei den einzelnen generierten Rezensionen nun die semantische Ähnlichkeit mit den Eingabebewertungen miteinbezogen. Hierdurch können auch generierte Bewertungen mit weniger überlappenden N-Grammen aber einer hohen semantischen Ähnlichkeit mit einer höheren Wahrscheinlichkeit ausgewählt werden.

### 7.6 Hyperparameter Optimierung mittels Dev-Datensatz

Die Hyperparameter $x, y, z, v$ der Rankingfunktion 28 legen bei der Auswahl der generierten Rezensionen fest, wie stark der ROUGE-1, ROUGE-2, ROUGE-L oder Moverscore berücksichtigt werden soll. Ein weiterer wichtiger Hyperparameter ist die Schrittweite $\alpha$ des Attributmodells. Die Schrittweite bestimmt den Grad der Korrektur des Gradienten pro Iteration des Attributmodells. Da die Datensätze jeweils unterschiedliche Eigenschaften haben, wie in Abschnitt 5 erläutert, bietet es sich an für die jeweiligen Datensätze eigene Hyperparameter zu finden.

Mit diesen adaptierten Rankingfunktionen ergeben sich bei der Auswertung auf dem Dev-Datensatz folgende in Tabelle 7.6 dargestellte Ergebnisse. Die Gewichtungen wurden unter Verwendung eines Greedy-Search-Algorithmus bestimmt. Für jede Schrittweite wurden jeweils die besten bestimmten Gewichtungen ausgewählt.

| DEV-Dataset | Amazon | | | | Yelp | | | |
|---|---|---|---|---|---|---|---|---|
| **Method** | **R1** | **R2** | **RL** | **MV** | **R1** | **R2** | **RL** | **MV** |
| *COOP + Attribute Model* | Stepsize $\alpha = 0.05$ | | | | | | | |
| Optimus | 36.31 | 7.17 | 20.75 | 56.82 | | | | |
| BIMEANVAE | 37.93 | 7.19 | 21.85 | 56.77 | | | | |
| *COOP + Attribute Model* | Stepsize $\alpha = 0.1$ | | | | | | | |
| Optimus | 36.40 | 7.78 | 20.99 | 56.78 | | | | |
| BIMEANVAE | **38.16** | 7.30 | <u>22.00</u> | 56.78 | | | | |
| *COOP + Attribute Model* | Stepsize $\alpha = 0.3$ | | | | | | | |
| Optimus | 35.03 | 6.81 | 19.44 | 56.22 | | | | |
| BIMEANVAE | 37.98 | **7.56** | 21.95 | <u>56.92</u> | | | | |
| *COOP + Attribute Model* | Stepsize $\alpha = 0.5$ | | | | | | | |
| Optimus | 35.84 | 6.93 | 19.75 | 56.23 | | | | |
| BIMEANVAE | <u>37.99</u> | <u>7.44</u> | **22.24** | **57.00** | | | | |
| *COOP* | | | | | | | | |
| Optimus | 35.97 | 7.16 | 20.15 | 23.22 | 35.50 | 7.84 | 19.26 | 23.33 |
| BIMEANVAE | 35.67 | 6.53 | 21.07 | 22.12 | 36.16 | 7.25 | 20.09 | 23.78 |

Tabelle 4: ROUGE und Moverscore Ergebnisse auf den DEV-Benchmarkdatensätzen für das COOP Modell und das optimierte COOP Attribut-Modell. Die besten Ergebnisse je Modellgruppe sind fett markiert und die zweitbesten Ergebnisse unterstrichen. $^\star$ denotiert, dass die ROUGE-Ergebnisse aus den Ergebnissen von (Iso et al., 2021) übernommen wurden.

Die in Tabelle 7.6 dargestellten Ergebnisse zeigen die Performance des kombinierten COOP + Attribut Modell im Vergleich zum normalen COOP Modell auf den Dev-Datensätzen. Die Ergebnisse zeigen eine Leistungssteigerung durch Verwendung des Attribut-Modells. Eine genaue Auswertung und Evaluierung findet in Abschnitt 8.3 auf dem Test-Datensatz statt.

Folgende in Tabelle 5 dargestellte Hyperparameter erzielen auf den Dev-Datensätzen die besten Ergebnisse und werden so in der späteren Evaluierung auf dem Test-Datensatz

verwendet.

| | Amazon | | | | | Yelp | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $\alpha$ | R1 (x) | R2 (y) | RL (z) | MV(v) | $\alpha$ | R1 (x) | R2 (y) | RL (z) | MV(v) |
| Optimus | - | | | | | - | | | | |
| BiMeanVAE | - | 3.5 | 0.0 | 1.0 | 0.5 | - | | | | |

Tabelle 5: Gewichtung der einzelnen Metriken zur Bestimmung des Input-Output-Overlap gesamt Scores

# 8 Evaluierung der Modelle

Die unterschiedlichen Optimierungen der Latentvektoren für ein Optimus Modell und des Cellstates für BIMEANVAE werden auf dem Amazon und dem Yelp Datensatz verglichen und mit dem aktuellem State-of-the-Art Modell COOP in Relation gesetzt. Es existieren im Dev-Datensatz und Test-Datensatz jeweils 8 Eingabebewertungen und drei Gold-Summaries zum Evaluieren der generierten Ausgabebewertungen. Die Eingabebewertungen werden durch ein Optimus VAE Modell und ein BIMEANVAE Modell in Latentvektoren umgewandelt. Anschließend werden mittels in Abschnitt 6.1 erklärter COOP Herangehensweise die Latentvektoren kombiniert, um den *Input-Output-Overlap* zu maximieren. Weiterhin werden die Latentvektoren mittels in dieser Arbeit eingeführtem Attribut-Modell optimiert, um detailreichere und präzisere Durchschnittsrezensionen zu erhalten.

## 8.1 Evaluationsmetriken

Die generierten Textbewertungen werden mit den drei Gold-Summaries verglichen. Zum Vergleich der generierten Rezensionen wird die ROUGE (**R**ecall-**O**riented **U**nderstudy for **G**isting **E**valuation)-Metrik verwendet. Die ROUGE-N Metrik misst die Anzahl der übereinstimmenden N-Grams zwischen dem generierten Text und den Referenztexten. Ein N-Gram ist eine N-lange sequentielle Folge von Wörtern innerhalb der Texte.

Zur Bewertung der generierten Bewertungen werden die vorhergesagten Ergebnisse mit den korrekten Ergebnissen verglichen. Die Konfusionsmatrix in Abbildung 9 ist eine Wahrheitsmatrix, welche die Einteilung der vorhergesagten Ergebnisse ermöglicht. True Positive (TP) und True Negative (TN) sind von dem Modell korrekt vorhergesagte Ergebnisse, False Positive (FP) und False Negative (FN) ist eine Klasse von falsch vorhergesagten Ergebnissen.

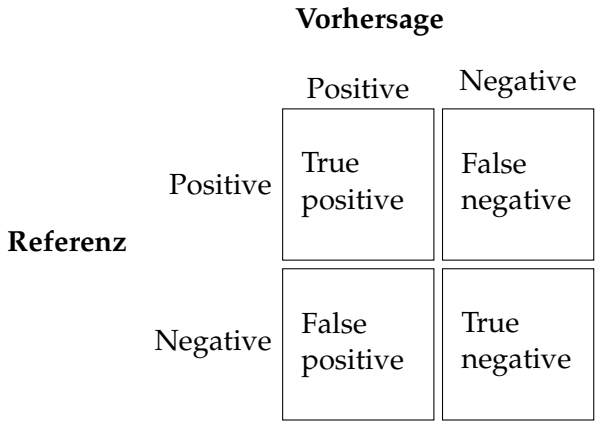



Abbildung 9: Konfusionsmatrix zur Berechnung des ROUGE-N Scores

Zur Berechnung des ROUGE-N Scores werden die einzelnen Textabschnitte in eine Menge aus N-Grams zerlegt. Mittels der Konfusionsmatrix in Abbildung 9 lassen sich Precision (P) und Recall (R) definieren:

**Precision**: Der Precision Wert ergibt sich aus dem Verhältnis der korrekt vorhergesagten N-Grams und der Anzahl der insgesamt vorhergesagten N-Grams.

$$P = \frac{TP}{TP + FP}$$

**Recall**: Recall ist als Verhältnis zwischen den korrekt vorhergesagten N-Grams und den N-Grams aus der Referenz definiert.

$$R = \frac{TP}{TP + FN}$$

**$F_1$**: Das F1-Maß beschreibt das harmonische Mittel zwischen Precision und Recall.

$$F_1 = \frac{2PR}{P + R}$$

In der Evaluation werden die ROUGE-1, ROUGE-2 und ROUGE-L Werte miteinander verglichen. ROUGE-1 verwendet als N-Gram Unigramme, ROUGE-2 Bigramme und ROUGE-L misst die längste gleiche Subsequenz zwischen Vorhersage und Referenz.

Da die ROUGE-Scores lediglich die einzelnen Wortsequenzen miteinander vergleichen, findet die semantische Bedeutung und Ähnlichkeit der Bewertungen mit der Referenz keinen Einfluss. Hier erzielen zum Beispiel Synonyme keine guten ROUGE-Scores, obwohl sie eine semantische Übereinstimmung haben. Um trotzdem die semantische Ähnlichkeit zwischen Bewertungen und Referenz zu messen, wird als weitere Metrik der Moverscore aus Abschnitt 8.2 verwendet. Der Moverscore basiert auf BERT und vergleicht Context-Embeddings mittels Earth-Mover-Distance. Als Metrik konnte der Moverscore hohe Korrelationen mit menschlichem Urteilsvermögen aufweisen.

## 8.2 Moverscore

Der Moverscore (Zhao et al., 2019) ist eine Evaluationsmetrik, die semantische Inhalte zwischen zwei Textsequenzen vergleicht und diesen einen Ähnlichkeitswert zuweist. Das Ziel vom Moverscore ist es eine Metrik abzubilden, die einer menschlichen Bewertung der Ähnlichkeit von zwei Sequenzen am nähesten ist. Im Gegensatz zu anderen Textähnlichkeitsmetriken, die lediglich die Überlappungen von Tokens innerhalb der Sequenzen messen ohne die Semantik der Wörter zu bewerten, bildet sich der Moverscore aus einer Kombination bestehend aus einer im Kontext eingebetteten Repräsentation der einzelnen Textsequenzen, die eine semantische Distanz untereinander abbilden. Die semantische Distanz wird über die Word Mover Distance (Kusner et al., 2015), einer Metrik basierend auf der Earth Mover Distance, bestimmt. Es wird ein minimaler Transportfluss zwischen den einzelnen Sequenzen errechnet. Die Worteinbettungen werden durch ein BERT Modell erzeugt.

Insgesamt ist der Moverscore für die Bewertungsgenerierung ein wichtiger Leistungsindikator, da nicht nur übereinstimmende N-Gramme an Wörtern gemessen werden,

sondern die Semantik der einzeln Wörter miteinbezogen wird. Da insbesondere in Bewertungen ähnliche Meinungen auf unterschiedliche Weise ausgedrückt werden können, bietet sich der Moverscore hier gut als Metrik an um diese Übereinstimmungen zu finden.

## 8.3 Ergebnisse

In Tabelle 8.3 ist die Performance der unterschiedlichen untersuchten und erstellten Modelle dargestellt. Das in dieser Arbeit entwickelte Modell ist das „COOP+Attribute Model“ Modell. Es basiert auf dem COOP Modell und verbessert die Generation von neuen Bewertungen durch die Verwendung eines Attribut-Modells. Unterschieden werden die COOP Modelle durch ihre grundlegend verwendete Variational Autoencoder Architektur in Optimus und BiMeanVAE. Das Optimus Modell kombiniert BERT und GPT-2 in ein Variational Autoencoder Modell. BiMeanVAE hingegen besteht aus einem BiLSTM Encoder mit einem LSTM Decoder trainiert als Variational Autoencoder Modell.

Verglichen wird die Performance der unterschieldichen Modelle mit den zuvor beschriebenen Evaluationsmetriken, dem ROUGE-1, ROUGE-2, ROUGE-L Score und dem Moverscore. Diese Metriken lassen ausreichend Rückschlüsse auf die erreichte Performance der Modelle und einer Leistungssteigerung zwischen den COOP Basismodellen und den modifizierten COOP mit Attributionsmodellen zu.

| Test-Dataset | Amazon | | | | Yelp | | | |
|---|---|---|---|---|---|---|---|---|
| **Method** | **R1** | **R2** | **RL** | **MV** | **R1** | **R2** | **RL** | **MV** |
| *COOP + Attribute Model* | | | | | | | | |
| Optimus | **37.01** | 7.44 | 20.55 | **23.86** | **35.99** | **7.79** | 19.40 | 23.56 |
| BiMeanVae | 36.47 | **7.59** | **22.22** | 23.05 | | | | |
| *COOP* | | | | | | | | |
| Optimus | 33.60 | 6.63 | 20.87 | 20.85 | 33.60 | 7.00 | 18.95 | 23.33 |
| BiMeanVae | 36.40 | 7.16 | 21.08 | 22.87 | 35.37 | 7.35 | **19.94** | 23.78 |
| *SimpleAvg* | | | | | | | | |
| Optimus * | 33.54 | 6.18 | 19.34 | 22.35 | 31.23 | 6.48 | 18.27 | 23.05 |
| BiMeanVae* | 33.60 | 6.64 | 20.87 | 20.85 | 32.87 | 6.93 | 19.89 | 22.41 |
| CopyCat * | 31.97 | 5.81 | 20.16 | - | 29.47 | 5.26 | 18.09 | - |
| MeanSum * | 29.20 | 4.70 | 18.15 | - | 28.46 | 3.66 | 15.57 | - |
| *Extractive* | | | | | | | | |
| LexRank * | 28.74 | 5.47 | 16.75 | - | 25.01 | 3.62 | 14.67 | - |

Tabelle 6: ROUGE und Moverscore Ergebnisse auf den Test-Benchmarkdatensätzen der unterschiedlichen Modelle. Die besten Ergebnisse sind fett markiert und die zweitbesten Ergebnisse unterstrichen. * denotiert, dass die ROUGE-Ergebnisse aus den Ergebnissen von (Iso et al., 2021) übernommen wurden.

Grundsätzlich lassen sich die Ergebnisse in Tabelle 8.3 in die Kategorien abstraktive und extraktive Zusammenfassung unterteilen. Hier ist eindeutig zu erkennen, dass LexRank

als extraktive Methode in allen Bereichen den abstraktiven Methoden unterliegt.

Die *SimpleAvg* Gruppe umfasst abstraktive Textzusammenfassungsmethoden. Bei dieser Gruppe wird von allen erzeugten Latentvektoren ein normaler Durchschnittsvektor errechnet, von dem anschließend gesamplet wird. Es ist erkennbar, dass die beiden Methoden Optimus und BIMEANVAE den Methoden CopyCat und MeanSum überlegen sind, da diese in allen Messwerten bessere Ergebnisse erzielen. Besonders hervozuheben ist hier, dass Optimus bessere Moverscore Ergebnisse als BIMEANVAE erzielt, allerdings in den ROUGE Werten minimal schlechter abschneidet.

Eine große Leistungssteigerung ergibt sich durch die COOP Methode, um eine optimale Kombination der einzelnen Latentvektoren zu finden. Hier erzielen sowohl Optimus wie auch BIMEANVAE in allen Metriken bessere Ergebnisse als die *SimpleAvg* Vergleichsgruppe. Demnach ist das Durchsuchen der Kombinationen von Latentvektoren sinnvoll. Insbesondere der ROUGE-1 Score übertrifft die *SimpleAVG* Scores bei Optimus und BIMEANVAE signifikant im Durchschnitt um 1.93%. Die größte Leistungssteigerung zwischen der COOP Kombinationsstrategie und *SimpleAvg* erfährt BIMEANVAE. Dies ist sehr beeindruckend, da *BiMeanVAE* mit 13 Millionen Parametern weitaus weniger Parameter hat als Optimus mit 239 Millionen Paramtern und auch nicht auf vortrainierte Sprachmodelle zurückgreifen kann. Demnach lassen sich mittels Variational Autoencoder Textsequenzen hervorragend in Latentvektoren encodieren und diese mittels Vektoroperationen kombinieren.

Eine weitere Leistungssteigerung der verwendeten Variational Autoencoder Optimus und BIMEANVAE lässt sich durch die in dieser Arbeit verwendeten Attributmodelle feststellen. Die aus dem Latentvektoren decodierten Textsequenzen lassen sich so noch stärker in die gewünschte Richtung bei der Generation adaptieren. Das verwendete Attributmodell ist ein Bag of Words Modell, welches aus den 150 am häufigsten vorkommenden Tokens besteht. Somit wird der Fokus bei der Generation auf die häufig vorkommenden Tokens gelegt und die generierten Textsequenzen haben eine höhere Überlappung. Beispielsweise fällt auf, dass beim Generieren teilweise mehrere vorgeschlagene Tokens ein hohes Ranking erhalten, wobei am Ende durch das Attributionsmodell das am besten passende mit einer höheren Wahrscheinlichkeit ausgewählt wird.

Die Performance des kombinierten COOP + Attributmodells zeigt in den ROUGE-1 Scores

## 8.4 Textgenerationsbeispiele

Folgend werden einige generierte Textrezensionen des COOP Modells und des COOP + Attributionsmodell Modells direkt miteinander verglichen. Zur besseren Darstellung sind bei den generierten Rezensionen die mit der Gold-Zusammenfassung übereinstimmenden Unigrams rot markiert, Bigrams sind farblich hinterlegt und die längste übereinstimmende Textsequenz unterstrichen. Der Moverscore lässt sich nicht visuell darstellen.

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

**Produkt:** NuGo Protein Bar, Vanilla Yogurt

**COOP+Attribute-Model:** easily orange easily background Apart from the obvious nutritional value, the taste was surprisingly better than I anticipated. Normally, if I find soy-based nutritional bars tend to have a chalky or tree-bark-like texture and taste. I dare say, the texture reminds me of a healthy-not-too-sweet 'Rice Krispy Treat'.
**Scores:** R-1: ,R-2: ,R-L: ,MV:

**COOP:** Apart from the obvious nutritional value, the taste was surprisingly better than I anticipated. Normally, if I find soy-based nutritional bars tend to have a chalky or tree-bark-like texture and taste. I dare say, the texture reminds me of a healthy-not-too-sweet 'Rice Krispy Treat'.
**Scores:** R-1: ,R-2: ,R-L: ,MV:

Abbildung 10: Vergleich der Generierten Rezensionen zu einem Produkt des Amazon Datensatzes

sanctus est Lorem ipsum dolor sit amet. 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.

**Restaurant Id:** 87YsVbCN_kfzheY79Fzjkg
**Review:** Great experience! Went cake tasting for our wedding and Jessica was a huge help!! She brought out the tastings for us after giving us time to look through their amazing book. She explained every cake and filling and had great recommendations for mixing the fillings. The champagne cake with cream cheese filling was amazing!! Chose our cake and design within an hour. Great service and knowledge. Thank you Jessica!

Abbildung 11: Vergleich der Generierten Rezensionen zu einem Restaurant des Yelp Datensatzes

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, 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.

# 9 Zusammenfassung und Ausblick

In dieser Masterarbeit konnte ein umfassender Überblick über State-of-the-Art Multi-Document Summarization im Bereich von Bewertungen gegeben werden.

## Literatur

Dor Bank, Noam Koenigstein und Raja Giryes (2020). „Autoencoders“. In: *CoRR* abs/2003.05991. arXiv: 2003.05991.

Arthur Brazinskas, Mirella Lapata und Ivan Titov (2019). „Unsupervised Multi-Document Opinion Summarization as Copycat-Review Generation“. In: *CoRR* abs/1911.02247. arXiv: 1911.02247.

Arthur Bražinskas, Mirella Lapata und Ivan Titov (Juli 2020). „Unsupervised Opinion Summarization as Copycat-Review Generation“. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, S. 5151–5169.

Ricardo Campos, Vitor Mangaravite, Arian Pasquali, Alipio Jorge, Celia Nunes und Adam Jatowt (2020). „YAKE! Keyword extraction from single documents using multiple local features“. In: *Information Sciences* 509, S. 257–289.

Eric Chu und Peter Liu (Juni 2019). „MeanSum: A Neural Model for Unsupervised Multi-Document Abstractive Summarization“. In: *Proceedings of the 36th International Conference on Machine Learning*. Hrsg. von Kamalika Chaudhuri und Ruslan Salakhutdinov. Bd. 97. Proceedings of Machine Learning Research. PMLR, S. 1223–1232.

Sumanth Dathathri, Andrea Madotto, Janice Lan, Jane Hung, Eric Frank, Piero Molino, Jason Yosinski und Rosanne Liu (2019). „Plug and Play Language Models: A Simple Approach to Controlled Text Generation“. In: *CoRR* abs/1912.02164. arXiv: 1912.02164.

Jacob Devlin, Ming-Wei Chang, Kenton Lee und Kristina Toutanova (2018). „BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding“. In: *CoRR* abs/1810.04805. arXiv: 1810.04805.

Günes Erkan und Dragomir R. Radev (2011). „LexRank: Graph-based Lexical Centrality as Salience in Text Summarization“. In: *CoRR* abs/1109.2128. arXiv: 1109.2128.

Hao Fu, Chunyuan Li, Xiaodong Liu, Jianfeng Gao, Asli Celikyilmaz und Lawrence Carin (2019). „Cyclical Annealing Schedule: A Simple Approach to Mitigating KL Vanishing“. In: *CoRR* abs/1903.10145. arXiv: 1903.10145.

Sepp Hochreiter und J¸rgen Schmidhuber (Dez. 1997). „Long Short-term Memory“. In: *Neural computation* 9, S. 1735–80.

Hayate Iso, Xiaolan Wang, Yoshihiko Suhara, Stefanos Angelidis und Wang-Chiew Tan (Nov. 2021). „Convex Aggregation for Opinion Summarization“. In: *Findings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Jeremy Jordan (Juli 2018). *Variational autoencoders.*

Diederik P Kingma und Max Welling (2014). *Auto-Encoding Variational Bayes*. arXiv: 1312.6114 [stat.ML].

Diederik P. Kingma und Max Welling (2019). „An Introduction to Variational Autoencoders“. In: *CoRR* abs/1906.02691. arXiv: 1906.02691.

Matt Kusner, Yu Sun, Nicholas Kolkin und Kilian Weinberger (Juli 2015). „From Word Embeddings To Document Distances“. In: *Proceedings of the 32nd International Conference on Machine Learning*. Hrsg. von Francis Bach und David Blei. Bd. 37. Proceedings of Machine Learning Research. PMLR, S. 957–966.

Chunyuan Li, Xiang Gao, Yuan Li, Xiujun Li, Baolin Peng, Yizhe Zhang und Jianfeng Gao (2020). „Optimus: Organizing Sentences via Pre-trained Modeling of a Latent Space“. In: *CoRR* abs/2004.04092. arXiv: 2004.04092.

Tomás Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado und Jeffrey Dean (2013). „Distributed Representations of Words and Phrases and their Compositionality“. In: *CoRR* abs/1310.4546. arXiv: 1310.4546.

Christopher Olah (o. D.). *Understanding LSTM networks*.

Jeffrey Pennington, Richard Socher und Christopher D Manning (2014). „Glove: Global Vectors for Word Representation.“ In: *EMNLP*. Bd. 14, S. 1532–1543.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee und Luke Zettlemoyer (2018). „Deep contextualized word representations“. In: *CoRR* abs/1802.05365. arXiv: 1802.05365.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei und Ilya Sutskever (2019). „Language Models are Unsupervised Multitask Learners“. In.

Rico Sennrich, Barry Haddow und Alexandra Birch (2015). „Neural Machine Translation of Rare Words with Subword Units“. In: *CoRR* abs/1508.07909. arXiv: 1508.07909.

Ilya Sutskever, Oriol Vinyals und Quoc V. Le (2014). „Sequence to Sequence Learning with Neural Networks“. In: *CoRR* abs/1409.3215. arXiv: 1409.3215.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser und Illia Polosukhin (2017). „Attention Is All You Need“. In: *CoRR* abs/1706.03762. arXiv: 1706.03762.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes und Jeffrey Dean (2016). „Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation“. In: *CoRR* abs/1609.08144. arXiv: 1609.08144.

Wei Zhao, Maxime Peyrard, Fei Liu, Yang Gao, Christian M. Meyer und Steffen Eger (2019). „MoverScore: Text Generation Evaluating with Contextualized Embeddings and Earth Mover Distance“. In: *CoRR* abs/1909.02622. arXiv: 1909.02622.

# Abbildungsverzeichnis



# Tabellenverzeichnis