Hochschule für Angewandte Wissenschaften Hamburg
*Hamburg University of Applied Sciences*

# Bachelorarbeit

Julian Mechow

Paraphrasierung von domänenspezifischen Texten
mit künstlichen neuronalen Netzen

*Fakultät Technik und Informatik*
*Department Informatik*

*Faculty of Engineering and Computer Science*
*Department of Computer Science*

**Julian Mechow**

# Paraphrasierung von domänenspezifischen Texten mit künstlichen neuronalen Netzen

**Bachelorarbeit** eingereicht im Rahmen der Bachelorprüfung

im Studiengang Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Olaf Zukunft
Zweitgutachter : Prof. Dr. Klaus-Peter Kossakowski

Abgegeben am 17.05.2019

**Julian Mechow**

**Thema der Bachelorarbeit**

Paraphrasierung von domänenspezifischen Texten mit neuronalen Netzen

**Stichworte**

Maschinelles Lernen, Computerlinguistik, Deep Learning, Seq2seq, Verarbeitung von natürlicher Sprache, Generierung von natürlicher Sprache, künstliche neuronale Netze

**Kurzzusammenfassung**

Als Paraphrasierung wird in der Kommunikationstheorie der Vorgang bezeichnet, in dem die Botschaft eines Textes in anderen Worten wiedergegeben wird. Dabei ist es wichtig, dass die zugrunde liegende Aussage nicht verändert wird. Die Computerlinguistik ist ein Forschungsgebiet aus dem Bereich der künstlichen Intelligenz, welche sich mit der Verarbeitung natürlicher Sprache im Kontext computerbasierter Berechnungen auf Basis von Algorithmen befasst. Dieser Forschungsbereich erhielt nicht zuletzt aufgrund von innovativen Lösungsansätzen in Bereichen wie unter anderem dem maschinellen Übersetzen natürlicher Sprachen, oder dem Entwickeln komplexer Dialogsysteme in der Vergangenheit zunehmend Aufmerksamkeit. Das maschinelle Paraphrasieren von Texten ist eine weitere Teildisziplin der Computerlinguistik und dient unter anderem dem automatisierten Generieren von alternativen Texten, oder dem Identifizieren von Paraphrasen im Zuge von Plagiatsprüfungen. In dieser Arbeit soll ein Einblick in die Verarbeitung natürlicher Sprache mit neuronalen Netzen im Kontext computerlinguistischer Verfahren gegeben werden. Im Wesentlichen wird hierbei auf das Lernverhalten neuronaler Netze, sowie auf deren Anwendung auf Probleme aus dem Raum der maschinellen Verarbeitung natürlicher Sprache eingegangen. Im praktischen Teil der Arbeit erfolgt die Anwendung der Verfahren in einem System zur Paraphrasierung von Texten aus der Wetterdomäne.

**Julian Mechow**

**Title of the paper**

Paraphrasing of domain-specific texts with neural networks

**Keywords**

Machine learning, natural language processing, deep learning, seq2seq, natural language generation, artificial neural networks

**Abstract**

Paraphrasing is the term used in communication theory to describe the message of a text in other words. It is important that the underlying information is not changed. Computational linguistics is a research field of artificial intelligence, which deals with the processing of natural language in the context of computer-based calculations based on algorithms. This field of research received increasing attention not least because of innovative solutions in areas such as the machine translation of natural languages, or the development of complex chatbots in the past. The automated paraphrasing of texts is another sub-discipline of computational linguistics and serves among other things the automated generation of alternative texts, or the identification of paraphrases in the course of plagiarism tests. In this work, an insight into the processing of natural language with neural networks in the context of computer linguistic procedures is given. Essentially, the learning behavior of neural networks as well as their application to problems in the field of machine processing of natural language are discussed. In the practical part of the thesis, the procedures are applied in a system for paraphrasing texts from the weather domain.

# Inhaltsverzeichnis

# 1 Einleitung

Diese Arbeit befasst sich mit der Entwicklung eines Systems zum automatisierten Generieren von Paraphrasen für Texte einer spezifischen Domäne. In diesem Kapitel werden Motivation, Zielsetzung und der Aufbau dieser Arbeit erläutert.

## 1.1 Motivation

In der heutigen Zeit gibt es unzählige Online-Plattformen, welche kontinuierlich neue Inhalte publizieren und jederzeit von nahezu überall abgerufen werden können. Vor allem News-Plattformen wie die Webseiten von Nachrichtendiensten, Finanzportalen oder Magazinen sind täglich auf umfangreiche Mengen von aktuellen Inhalten angewiesen, dabei ist die Nachfrage nach hochwertigen Texten besonders hoch. Die Qualität, die Popularität und die Aktualität der bereitgestellten Texte korrelieren mit der Anzahl der Aufrufe einer Webseite. Eine hohe Anzahl an Seitenaufrufen bedeutet in der Regel auch mehr Geld für die Betreiber der Online-Plattform. [Mey18]

Aufgrund des hohen Bedarfs an Textinhalten ist die Nachfrage nach maschinell generierten Texten, welche jederzeit abgerufen werden können sehr hoch. Die Vorteile liegen vor allem in der Flexibilität dieser Lösung und der Einsparung von Personal, welches die benötigten Texte andernfalls manuell verfassen müsste. Die Computerlinguistik ist ein populärer Forschungsbereich, der zusammen mit den Anfängen der künstlichen Intelligenz in den 1960er Jahren entstand. Das Ziel dieses Teilbereiches der künstlichen Intelligenz ist es natürliche Sprachen in Wort und Schrift zu formalisieren und diese in einer für Computersysteme verständlichen Form zu verarbeiten, um einen natürlichen Kommunikationskanal zwischen Menschen und Maschinen zu entwickeln. In der heutigen Zeit ist es mit Hilfe leistungsstarker Hardware möglich die benötigte Rechenleistung aufzubringen, um Konzepte und Theorien der künstlichen Intelligenz in einer Vielzahl Anwendungsszenarien zu nutzen, um komplexe Probleme durch die maschinelle Verarbeitung natürlicher Sprachen zu lösen. Im Zuge dessen entstanden unter anderem

intelligente Übersetzungssysteme, Systeme zum Verarbeiten und Extrahieren von Informationen aus Texten, sowie Systeme zum Klassifizieren von Texten und Dokumenten oder auch Systeme zum automatisierten Generieren von Text auf Grundlage angelernter Informationen. [Khu17]

Das maschinelle Paraphrasieren von Text ist eine fundamentale Teilaufgabe verschiedener Anwendungen der Computerlinguistik. Zu den bekanntesten Problemen aus dem Bereich der maschinellen Paraphrasierung gehören (1) das Identifizieren von Paraphrasen eines gegebenen Textes, (2) das Extrahieren von Paraphrasen und (3) die Generierung von Paraphrasen zu einem gegebenen Text. Diese Arbeit konzentriert sich auf die Generierung von Paraphrasen mit dem Fokus auf die sequenzielle Verarbeitung natürlicher Sprache und den zugrundeliegenden Konzepten und Theorien der künstlichen Intelligenz. [Pra16]

## 1.2 Zielsetzung

Das Ziel dieser Arbeit ist es Paraphrasen für Texte einer spezifischen Domäne zu generieren. Dieses Vorhaben wird mit Hilfe neuronaler Netze im Kontext der sequenziellen Verarbeitung natürlicher Sprache, mit dem Ziel eine Eingabesequenz in eine gewünschte Ausgabesequenz zu überführen in die Tat umgesetzt. Das Hauptaugenmerk liegt hierbei auf der Generierung von alternativen Texten, welche mit einer möglichst hohen Textvarianz die gleichen Informationen abbilden, wie der Text der Eingabesequenz und dabei die semantischen und syntaktischen Anforderungen der verwendeten Sprache erfüllen. Das im Zuge dieser Bachelorarbeit entwickelte System dient der Paraphrasierung von deutschen Wetterberichten und soll treffende Alternativen zu gegebenen Wettertexten erzeugen. Um dieses Ziel zu erreichen wird für das entwickelte System mit einer Vielzahl von deutschsprachigen Wettertexten trainiert. Die Beschaffung und die Aufbereitung der Trainingsdaten ist eine wesentliche Teilaufgaben mit der sich der Inhalt dieser Arbeit befasst. Das System könnte im späteren Verlauf auch auf Texten anderer Domänen trainiert werden und beispielsweise Plattformen automatisiert mit Inhalten beliefern. Die Kernaufgabe dieser Arbeit ist die Untersuchung der sequenziellen Verarbeitung und Generierung von natürlich sprachlichen Texten, unter der Verwendung von neuronalen Netzen. Hierbei wird die grundlegende Idee und Funktionsweise von neuronalen Netzen erörtert und es wird untersucht wie diese sich zur Lösung des gegebenen Problems nutzen lassen. Im praktischen Teil der Arbeit wird auf Basis der gewonnenen Erkenntnisse ein System zur Lösung der gegebenen Problemstellung entwickelt. Die Dokumentation des Entwicklungsprozesses und die praktische Anwendung der untersuchten Konzepte, Frameworks und Technologien ist ein weiterer wichtiger Bestandteil, auf den sich der Inhalt dieser Arbeit konzentriert. Abschließend werden verschiedene Experimente mit dem entwickelten System durchgeführt und die daraus gewonnenen Forschungsergebnisse werden im Zuge dieser Bachelorarbeit analysiert und ausgewertet.

## 1.3 Gliederung

Der Inhalt dieser Arbeit erstreckt sich über mehrere Kapitel, dessen Inhalt in diesem Abschnitt kurz erläutert wird.

### 1.3.1 Grundlagen

Dieses Kapitel enthält die notwendigen Grundlagen, welche für das Verständnis und die Realisierung des angestrebten Systems benötigt wird.

### 1.3.2 Problemstellung

In diesem Kapitel wird das zu lösende Problem erläutert und es wird eine Anforderungsanalyse durchgeführt, auf deren Basis im Anschluss die Spezifikation für das zu entwickelte System definiert wird.

### 1.3.3 Konzept

Aufbauend auf der Spezifikation des Systems werden in diesem Kapitel unter Einsatz von wissenschaftlichen Methoden des Software-Engineerings der konzeptionelle Aufbau und das Design des Systems entwickelt. In dem Kapitel werden die getroffenen Entwurfs-entscheidungen, sowie die Systemarchitektur dokumentiert. Des Weiteren werden Maßnahmen zur Qualitätssicherung erörtert.

### 1.3.4 Realisierung

Dieses Kapitel beschäftigt sich mit der Implementierung des angestrebten Systems und erläutert die Funktionalität und den Aufbau der einzelnen Systemkomponenten.

### 1.3.5 Experimente und Auswertung

Die Ergebnisse dieser Arbeit werden in diesem Kapitel vorgestellt, untersucht und in Relation miteinander gesetzt. Zunächst werden die Ergebnisse des Basisversuchs analysiert. Auf diesem Versuch aufbauen werden anschließend verschiedene Experimente zur Systemoptimierung durchgeführt und analysiert.

## Fazit

Das letzte Kapitel enthält eine Zusammenfassung der Arbeit, sowie ein Fazit und einen Ausblick darauf, wie zukünftige Projekte an die Ergebnisse dieser Bachelorarbeit anknüpfen könnten.

# 2 Grundlagen

In diesem Kapitel werden die zugrundeliegenden Konzepte und Methoden behandelt, welche für die Realisierung des praktischen Teils dieser Bachelorarbeit eine tragende Rolle spielen. Es bietet eine Übersicht über die Terminologie des Themenbereichs der maschinellen Verarbeitung von natürlicher Sprache und grenzt die einzelnen Felder dieses Bereiches voneinander ab. Zudem enthält das Kapitel eine Einführung in die Funktionsweise von künstlichen neuronalen Netzen und stellt das verwendete Seq2Seq Framework vor.

## 2.1 Abgrenzung und Terminologie

Im Folgenden werden die Begrifflichkeiten Natural Language Processing (NLP), Natural Language Generation (NLG) und Natural Language Understanding (NLU), welche häufig im Kontext computerlinguistischer Systeme verwendet werden, vorgestellt und voneinander abgegrenzt. NLP dient dem maschinellen Verarbeiten natürlicher Sprache und stellt einen wichtigen Baustein bei der Entwicklung künstlicher Intelligenz im linguistischen Raum dar. Die zugrundeliegende Annahme hinter NLP baut darauf auf, dass jegliche Form von natürlicher Sprache erkannt und anschließend in einer für den Computer verständlichen Form extrahiert werden muss. Da es sich bei natürlicher Sprache um ein komplexes System handelt, können einzelne Worte nicht atomar betrachtet werden, sondern müssen in Zusammenhang mit weiteren Satzfragmenten analysiert werden, um die Bedeutung eines gegebenen Sprachkonstruktes zu rekonstruieren. Um dies zu erreichen verwenden NLP Systeme verschiedene Algorithmen aus dem Bereich des maschinellen Lernens. Diese dienen dazu Rohdaten in einer für die Verarbeitung notwendigen strukturierten Form abzubilden und ermöglichen es dem System anhand großer Trainingsdatensätze allgemeine Muster in den Daten zu erkennen. [Kum18]

Das daraus gewonnene Verständnis der Sprache wird als NLU bezeichnet. NLU ist die vom Computer interpretierte Bedeutung eines gegebenen Sprachkonstruktes. Dazu gehört das Klassifizieren von Satzgliedern, das Parsen einzelner Sätze und das Erstellen eines Kontextmodells. Eine wichtige Voraussetzung für NLU ist ein umfangreiches Lexikon, welches die Bedeutung der einzelnen, zum Teil auch ambivalenten Worte einer Sprache enthält. Das in Abbildung 1 dargestellte Venn-Diagramm legt offen, dass die Übergänge zwischen NLP und NLU fließend sind. Es lässt sich jedoch festhalten, dass NLP vornehmlich dazu dient, Texte in strukturierte Daten abzubilden und ihre Bedeutung zu extrahieren, während die Aufgabe von

NLU Prozessen darin liegt, die Bedeutung der Texte zu untersuchen und zu verstehen. [Kum18]

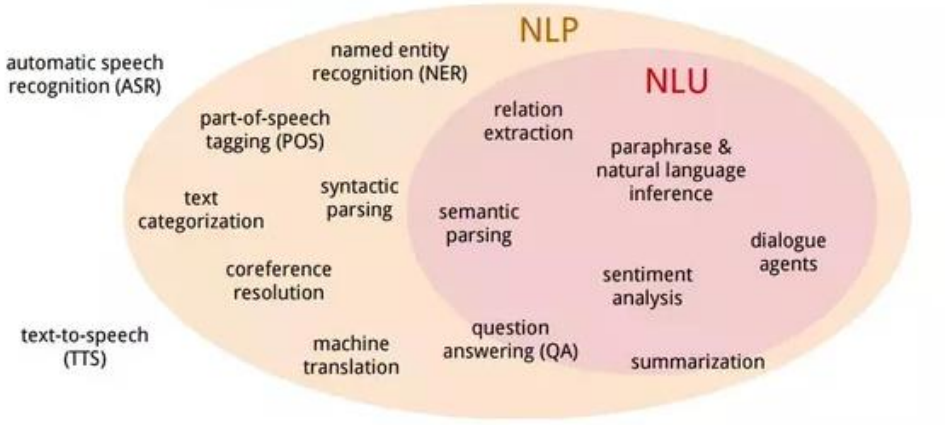



**Abbildung 1**: NLP vs. NLU (Quelle: [Mac16, S. 8])

NLG baut auf den Ergebnissen aus NLP und NLU auf und nutzt die strukturierten Daten und das gewonnene Verständnis über die Bedeutung des Inhalts, um daraus einen neuen, natürlich sprachlichen Text zu generieren. Hierbei interpretiert das System den gelernten Kontext, um neue Inhalte zu erzeugen, welche je nach Anwendungsfall und trainierten Modell variieren. NLG wird beispielsweise in Übersetzungssystemen verwendet, um Texte, welche mit NLP und NLU verarbeitet und analysiert wurden in eine Zielsprache zu übersetzen, indem der gelernte Kontext mit Hilfe eines auf der Zielsprache trainierten Modells interpretiert wird. [Kum18]

Zusammengefasst lässt sich sagen, dass NLP den Prozess definiert, indem ein Computer natürlich sprachliche Inhalte liest und strukturiert. NLU setzt auf dieser strukturierten Interpretation der Sprache auf und untersucht diese, um deren Bedeutung zu bestimmen. NLG Prozesse nutzen die dadurch gewonnenen Informationen um aus den strukturierten Daten, zusammen mit dem interpretierten Kontext neue Inhalte zu generieren. [Kum18]

## 2.2 Künstliche neuronale Netze

Es existieren komplexe Problemklassen, welche von einer Vielzahl subtiler Faktoren abhängig sind. Solche kontextsensitiven Problemstellungen entziehen sich oft dem Wirkungsbereich statischer Lösungsansätze, da diese häufig keine allgemeingültige Lösung für weitere Probleme dieser Art darstellen. In einer naiven Herangehensweise könnte man zu dem Schluss kommen, dass sich ein solches Problem durch die Entwicklung eines komplexen Systems aus statischen Regeln, welches alle möglichen Faktoren berücksichtigt lösen ließe.

In der Realität wäre ein solches System jedoch sehr unübersichtlich und äußerst schwer zu warten. Die Komplexität des Regelkonstrukts würde schnell ein exponentielles Wachstum erreichen und es ist unwahrscheinlich, dass dabei tatsächlich alle möglichen Szenarien abgedeckt werden. [Kie05]

Für genau diese Art von Problemklassen wurden künstliche neuronale Netze entwickelt. Hierbei handelt es sich um Rechenmodelle, welche von der Funktionsweise der neuronalen Netze des menschlichen Gehirns inspiriert wurden. Diese verfügen über wünschenswerte Eigenschaften zur Lösung komplexer Probleme, an denen es Computern trotz ihrer enormen Rechenleistung mangelt. Die Fähigkeit zu denken, Fehlentscheidungen zu erkennen, zu analysieren und anschließend reflektiv aus diesen Fehlern zu lernen und die daraus gewonnene Erfahrung in zukünftige Entscheidungsprozesse mit einfließen zu lassen. Um dieses Potential zu nutzen benötigen künstliche neuronale Netze eine große Anzahl an exemplarischen Trainingsdaten, basierend auf Lösungsbeispielen für Probleme einer bestimmten Klasse. Dieser Trainingsprozess ermöglicht es assoziativ Muster zu erkennen und generalisierte Lösungswege zu finden, ohne dass dabei statische Lösungen für spezifische Probleme gesucht werden. Daher ist es möglich auf Basis des erlernten Verhaltens auch andere Probleme derselben Problemklasse zu lösen. [Kie05]

### 2.2.1 **Aufbau**

Ein künstliches neuronales Netz besteht aus einer Vielzahl einfacher Recheneinheiten, welche analog zum biologischen Vorbild Neuronen genannt werden. Ähnlich wie bei den Neuronen eines Nervensystems erhalten diese künstlichen Neuronen bestimmte Reize, beziehungsweise Eingabesignale und senden ein Signal an weitere Neuronen, sobald ein bestimmter Schwellwert erreicht wurde. Ein Neuron stellt den Grundbaustein für das konnektionistische Modell eines neuronalen Netzes dar. Es besteht aus einer Gewichtung, einer Propagierungsfunktion, einer Aktivierungsfunktion und einem Schwellwert. [Kie05]

Abbildung 2 zeigt ein Neuron, welches die Eingabesignale $x_1, \dots, x_n$ erhält, diese können auch als Eingabevektor $X$ betrachtet werden. Zudem verfügen Neuronen über ein- und ausgehende, gerichtete Kanten, welche die Recheneinheiten miteinander vernetzen. Dabei verfügt jede dieser Kanten über eine eigene Gewichtung $w_{ij}$, die darüber entscheidet, wie groß der Einfluss eines auf dieser Kante gesendeten Signals von einem Neuron $i$ auf ein weiteres Neuron $j$ ist. Die in Abbildung 2 dargestellten Gewichte $w_{1j}, \dots, w_{nj}$ können auch als Gewichtsvektor $W$ interpretiert werden. Die Aufgabe der Propagierungsfunktion $\sum$ ist die Berechnung einer Netzeingabe $net_j$, welche mit Hilfe der Eingabesignale $X$ und der Kantengewichten $W$ berechnet wird. [Cas16]

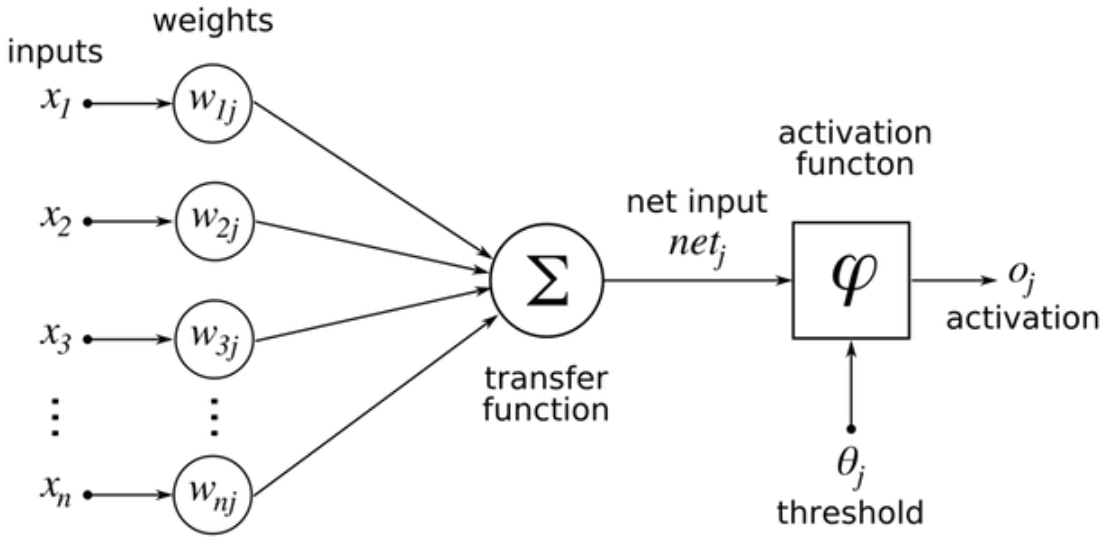



**Abbildung 2:** Datenverarbeitung eines Neurons (Quelle: [Cas16])

Die Netzeingabe für ein Neuron setzt sich zusammen aus der Summe der Produkte aller eingehenden Signale und den Gewichten der jeweiligen Kanten. Diese Berechnung erfolgt durch die Propagierungsfunktion, welche die Eingabesignale $x_{i_1}, \dots, x_{i_n}$ von anderen Neuronen $i_1, \dots, i_n$, die über ausgehende Kanten mit einem Neuron $i$ verknüpft sind unter Berücksichtigung der jeweiligen Gewichte $w_{ij}$ berechnet.

$$\sum_{i \in I} (x_i \cdot w_{ij})$$

Der daraus resultierende Netzeingabe wird anschließend mit einer Aktivierungsfunktion verrechnet. Die Aktivierungsfunktion stellt den Zusammenhang zwischen der Netzeingabe eines Neurons und dessen Aktivitätslevel dar. Erreicht das Ergebnis der Aktivierungsfunktion einen vom Neuron abhängigen Schwellwert $\Theta_j$, so wird das Ergebnis $o_j$ als Eingabesignal an alle über ausgehende Kanten verknüpften Neuronen gesendet. [Kie05]

Ein künstliches neuronales Feed-Forward-Netz setzt sich in den meisten Fällen aus mehreren Schichten zusammen. Dazu gehört der Input-Layer, eine endliche Menge von Hidden-Layers und der Output-Layer. Die Neuronen des Input-Layers erhalten Signale der Außenwelt in Form von numerischen Werten, welche bestimmte Merkmale und Eigenschaften repräsentieren. Diese werden häufig als Features bezeichnet und dienen als mathematisches Abbild von repräsentativen Eigenschaften der Problemdomäne. Dabei erhält jedes Neuron des Input-Layers die Daten eines bestimmten Features. Während der Input-Layer eine direkte Verbindung zur Außenwelt besitzt stellen die Hidden-Layers die netzinterne Interpretation der Außenwelt dar. Sie verarbeiten Informationen aus dem Input-Layer und übermitteln die daraus resultierenden Informationen an den Output-Layer des neuronalen

Netzes. Der Output-Layer sammelt die vom Netz verarbeiteten Informationen und ist für die Verdichtung der gesammelten Daten zu einem Ergebnisvektor zuständig. Zudem ist es seine Aufgabe dieses Ergebnis wieder an die Außenwelt zu übermitteln. Abbildung 3 zeigt den exemplarischen Aufbau eines künstlichen neuronalen Netzes und dessen Schichtenverteilung. [Ujj16]

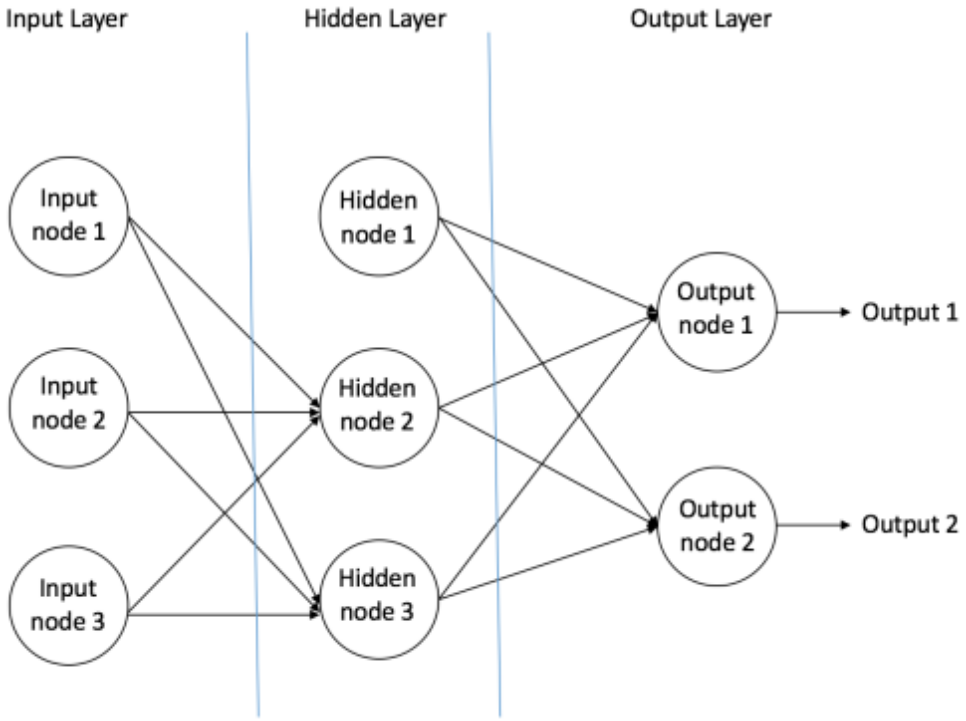



**Abbildung 3:** Ein Beispiel eines feed-forward neural networks (Quelle: [Ujj16])

### 2.2.2 **Lernprozess**

Beim maschinellen Lernen werden mehrere Lernverfahren unterschieden. Dazu gehören unter anderen das überwachte Lernen, das unüberwachte Lernen und das bestärkende Lernen. Beim überwachten Lernen werden Trainingsdaten verwendet, welche mit einem gewünschten Ergebniswert gekennzeichnet sind. Diese dienen dem neuronalen Netz als Maßstab für die Fehlerberechnung, um die Abweichung zwischen dem Sollwert und dem Istwert der Vorhersage zu ermitteln. Hierbei werden die Gewichte des Netzes vom Output-Layer bis hin zum Input-Layer iterativ anpasst, so dass sich der errechnete Fehler immer weiter reduziert. Beim unüberwachten Lernen gibt es keine Zielwerte in den Trainingsdaten, welche den gewünschten Ergebniswert zu einem Datensatz angeben. Hierbei geht es darum Zusammenhänge in den Daten zu finden und diese zu gruppieren. Die dabei eingesetzten Algorithmen dienen vornehmlich dazu große, unstrukturierte Datenmengen explorativ zu erkunden, um Gemeinsamkeiten und Muster in den Daten zu finden. Beim bestärkenden

Lernen wird ein sogenannter Agent verwendet, welcher mit Hilfe eines Belohnungs- und Bestrafungssystems selbstständig Lösungsstrategien zu situativen Problemen ermittelt. Die Inhalte dieser Arbeit beschränken sich auf das Lernverfahren des überwachten Lernens, welches im praktischen Teil der Arbeit angewendet wird. [Ujj16]

Der Lernprozess des überwachten Lernens benötigt eine große Menge an gekennzeichneten Trainingsdaten, um das Netz so zu optimieren, dass es in der Lage ist allgemeine Lösungsansätze für die Problemdomäne der eingespeisten Daten zu finden. Um eine generalisierte Lösung für eine Klasse von Problemen zu finden ist eine große Fehlertoleranz gegenüber den Daten notwendig, um ein hohes Maß an Korrektheit zu gewährleisten. Beispielsweise kann ein zu kleiner Trainingsdatensatz, oder eine mangelnde repräsentative Vielfalt in den Daten dazu führen, dass ein hoher Varianzwert entsteht, was zur Folge hat, dass das System eine Überanpassung im Lernalgorithmus vornimmt. Daraus resultiert, dass das System eine Übergeneralisierung vornimmt und einzelne Datenstrukturen auswendig lernt. Auf der anderen Seite können auch Verzerrungsfehler die Korrektheit des Systems gefährden. Die Ursache hierfür liegt meist bei der Auswahl von Berechnungsmodellen, oder den gewählten Features, welche für den Lernalgorithmus verwendet werden. Diese beiden Probleme werden auch Über- und Unteranpassung genannt und beziehen sich auf die Anpassung der Gewichte des Netzwerks, während des Trainings. [Gér17]

Das Training eines neuronalen Netzes durchläuft im überwachten Lernen eine endliche Anzahl von Epochen, in denen das Netz durch das iterative Anpassen der Gewichte schrittweise optimiert wird. Die Gewichte der einzelnen Kanten werden häufig mit zufälligen Werten initialisiert. Das Netz erhält einen Trainingsdatensatz als Eingabesignal über den Input-Layer, dieser wird in den Hidden-Layers verarbeitet und anschließend im Output-Layer gesammelt und verdichtet. Die daraus entstehende Netzausgabe wird mit dem entsprechenden Sollwert aus dem gekennzeichneten Datensatz verglichen und die daraus resultierende Abweichung stellt den Fehlerwert dar, der im Verlauf des Trainings reduziert werden soll. Ein bewährtes Maß zur Ermittlung des durchschnittlichen Fehlers des Systems ist der Root Mean Square Error (RMSE).

$$RMSE(X,h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(h(x^{(i)}) - y^{(i)})^2}$$

Diese Funktion aus der Statistik berechnet die Wurzel der mittleren quadratischen Fehlersumme des Systems für einen gegebenen Trainingsdatensatz. Hierbei wird $m$ durch die Anzahl der Instanzen in den Trainingsdaten bestimmt. Die Funktion $h\left(x^{(i)}\right)$ definiert die Hypothese, welche die vom System berechnete Vorhersage für einen Feature-Vektor $x^{(i)}$ darstellt. $x^{(i)}$ ist ein Vektor, welcher die Featurewerte für die $i^{th}$ Instanz des Trainingsdatensatzes enthält. $y^{(i)}$ stellt den jeweiligen Sollwert für die Vorhersage der $i^{th}$

Instanz dar. Das Ergebnis wird quadriert, um zu verhindern, dass sich negative und positive Abweichungen gegenseitig annullieren. Diese Funktion beschreibt, wie hoch die durchschnittliche Abweichung zwischen den getroffenen Vorhersagen und den jeweiligen Sollwerten ist. [Gér17]

RMSE wird häufig für Regressionsaufgaben verwendet, es gibt jedoch auch Fälle, in denen der absolute Fehlerwert unabhängig von der Richtung in der dieser von dem Sollwert abweicht relevant ist. Dafür wird häufig die Funktion Mean Absolute Error (MAE) verwendet. Diese misst die durchschnittliche Abweichung zwischen den Soll- und Istwerten. Sowohl MAE als auch RMSE dienen der Ermittlung der Abweichung zwischen zwei Vektoren, dem Vorhersagevektor und dem Vektor der entsprechenden Sollwerte. [Gér17]

$$MAE(X, h) = \frac{1}{m}\sum_{i=1}^{m}\left|h\left(x^{(i)}\right) - y^{(i)}\right|$$

Um den Fehlerwert eines neuronalen Netzes zu verringern ist es zunächst notwendig die Ursache für die Abweichung zwischen den vorhergesagten und den erwarteten Werten zu finden. Darauf aufbauend müssen die Gewichte der Kanten so angepasst werden, dass sich die Netzausgabe gegen den Sollwert konvergiert. Dieser Vorgang erfolgt durch das Konzept der Backpropagation. Hierbei berechnet das neuronale Netz die Netzausgabe für jede Trainingsinstanz und ermittelt den entsprechenden Fehlerwert. Anschließend wird berechnet, wie groß der Einfluss jedes Neurons aus der vorherigen Schicht auf das Ergebnis ist. Diese wiederum berechnen, wie groß der Einfluss der Neuronen ihrer Vorgängerschicht ist. Dieser Vorgang wiederholt sich, bis der Algorithmus den Input-Layer erreicht. Durch diese Rückverfolgung der Berechnung der Netzausgabe ist es möglich die Fehlergradienten für alle Kantengewichtungen zu ermitteln. Im letzten Schritt der Backpropagation wird mit Hilfe des Gradientenabstiegsverfahrens ein Optimierungsschritt auf Basis der gesammelten Gradienten auf allen Kantengewichten durchgeführt, um diese schrittweise zu optimieren. Das Gradientenabstiegsverfahren ist ein generischer Optimierungsalgorithmus, dessen Ziel es ist durch iteratives Anpassen der Modellparameter die Fehlerfunktion zu minimieren. Als Fehlerfunktion wird hierbei häufig Mean Square Error (MSE) statt Root Mean Square Error (RMSE) verwendet, da es in der Praxis häufig einfacher ist, diese zu minimieren und beides zum gleichen Ergebnis führt. $\theta$ ist hierbei der Vektor, die Gewichte enthält und $\theta^T$ transponiert diesen Vektor, so dass ein Zeilen- statt Spaltenvektor verwendet wird. [Gér17]

$$MSE(X, h_\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\theta^T \cdot x^{(i)} - y^{(i)}\right)^2$$

Beim Gradientenabstiegsverfahren wird der lokale Gradient der Fehlerfunktion betrachtet und dessen Parameter werden iterativ angepasst, so dass der Gradient immer weiter reduziert wird. Das Ziel ist es hierbei, dass der Gradient den Wert 0 erreicht und damit ein

Minimum findet. Hierbei hängt die Größe der Anpassungsschritte von der gewählten Lernrate $\eta$ ab. Die Auswahl einer geeigneten Lernrate ist ein wichtiger Bestandteil der Hyperparameteroptimierung. Falls die Lernrate zu gering ist, werden sehr viele Iterationen benötigt, um ein Minimum zu erreichen. Wenn die gewählte Lernrate zu hoch ist, fallen die Optimierungsschritte zu groß aus und es ist unwahrscheinlich, dass ein Minimum gefunden wird. [Gér17]

Um das Gradientenabstiegsverfahren zu verwenden müssen zunächst für alle Gewichte $\theta_j$ die Gradienten der Fehlerfunktion berechnet werden, um zu ermitteln wie groß der Einfluss der gewünschten Anpassung auf die Fehlerfunktion ist. Zu diesem Zweck wird die partielle Ableitung der Fehlerfunktion verwendet. [Gér17]

$$\frac{\partial}{\partial \theta_j} MSE = \frac{2}{m} \sum_{i=1}^{m} \left(\theta^T \cdot x^{(i)} - y^{(i)}\right) x_j^{(i)}$$

Die Berechnung der einzelnen Gradienten mittels partieller Ableitung kann wie folgt zusammengefasst werden:

$$\nabla_\theta MSE(\theta) = \begin{bmatrix} \frac{\partial}{\partial_0^\theta} MSE(\theta) \\ \frac{\partial}{\partial_1^\theta} MSE(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_n} MSE(\theta) \end{bmatrix} = \frac{2}{m} X^T \cdot (X \cdot \theta - y)$$

Mit Hilfe des daraus resultierenden Gradientenvektors kann ein Optimierungsschritt des Gradientenabstiegsverfahrens durchgeführt werden. Hierbei wird der Gradientenvektor mit der Lernrate $\eta$ multipliziert und anschließend vom Vektor der Gewichte subtrahiert.

$$\theta^{(next\ step)} = \theta - \eta \nabla_\theta MSE(\theta)$$

Dieser Schritt wird während der Backpropagation in jeder Epoche durchgeführt, um die Kantengewichte des Netzes zu optimieren und damit ein Modell zu trainieren, indem die Fehlerfunktion kontinuierlich minimiert wird. [Gér17]

### 2.2.3 Deep Learning

Deep Learning ist ein Begriff, der das Fachgebiet der künstlichen Intelligenz stark geprägt hat und im Kontext von Applikationen anzutreffen ist, welche auf künstlichen neuronalen Netzen basieren. Hierbei handelt es sich um ein solches Netzwerk, dessen Schichtenstruktur über mehrere Hidden-Layer verfügt, welche die Tiefe eines solchen Netzes definieren. Deep Learning Netze verfolgen das Ziel Gehirnprozesse zu simulieren, um Lernalgorithmen zu verbessern und ihre Benutzbarkeit zu optimieren. Die Grundvoraussetzung für solche komplexen Netze sind zum einen leistungsfähige Computer und zum anderen eine sehr große Menge an Daten, welche es ermöglichen, solche umfangreiche neuronale Netze zu trainieren. [Bro16]

Die Multi-Layer-Architektur solcher neuronalen Netze unterscheidet sich vor allem dadurch von anderen maschinellen Lernverfahren, dass die Performanz des Systems mit der Menge an Trainingsdaten skaliert. Ein weiterer entscheidender Vorteil von Deep Learning Technologien ist, dass sie in der Lage sind automatisiert Features zu extrahieren. Dieser Vorgang wird häufig auch als Feature Learning bezeichnet. Die Extraktion von Features ist eine aufwändige und sehr wichtige Teilaufgabe des Lernprozesses eines künstlichen neuronalen Netzes. Deep Learning Systeme trainieren in jedem Hidden-Layer unterschiedliche Features, basierend auf der Ausgabe der vorherigen Schicht. Durch diese Art der Informationsverarbeitung werden die Features in jedem weiteren Layer zunehmend komplexer. Während des Trainings lernen die einzelnen Schichten durch wiederholtes Kombinieren der Ergebnisse vorheriger Schichten Korrelationen zu erkennen und daraus neue repräsentative Features zu extrahieren. [Sky18]

### 2.2.4 Recurrent Neural Networks

Natürliche Sprache wird in der Regel zeichenweise oder wortweise in Sequenzen verarbeitet. Herkömmliche neuronale Netze mit einer Feed-Forward Architektur haben den Nachteil, dass diese nicht für die Verarbeitung sequenzieller Daten geeignet sind, da jede Eingabe einer solchen Sequenz in Relation zu den kommenden und vorrausgegangenen Eingaben stehen. Das Problem liegt darin, dass diese Netze keine Informationen über verarbeitete Eingaben speichern und sich die Informationen nur in eine Richtung bewegen, nämlich vom Input-Layer über die Hidden-Layers bis zum Output-Layer, daher verarbeitet ein Neuron eine bestimmte Information nur genau einmal. Für eine sequenzielle Verarbeitung ist es jedoch notwendig, dass die Informationen von bereits verarbeiteten Eingaben zu jedem Zeitschritt in die Verarbeitung der aktuellen Sequenzeingabe mit einfließen, um eine Vorhersage über die nächste Sequenzeingabe zu ermöglichen. [Don18]

Für die sequenzielle Verarbeitung natürlich sprachlicher Texte werden daher häufig besondere Varianten von neuronalen Netzen eingesetzt, mit dessen Hilfe es möglich ist bereits verarbeitete Informationen zu speichern und diese bestimmten Zeitschritten

zuzuordnen. Diese Netze werden als Recurrent Neural Networks (RNN) bezeichnet und sind dazu befähigt ein tieferes Verständnis für sequenzielle Daten zu entwickeln, indem sie Informationen zyklisch verarbeiten und bereits verarbeitete Daten in die Verarbeitung neuer Netzeingaben mit einbeziehen. Ein RNN erhält in jedem Zeitschritt zwei Eingaben, zum einen die aktuelle, zu verarbeitende Information und zum anderen das Ergebnis der vorherigen Netzausgaben, welche als Hidden-States bezeichnet werden. Darüber hinaus können RNNs anders als Feed-Forward Netze nicht nur eine Eingabe zu einer Ausgabe überführen, stattdessen ist es ihnen möglich eine endliche Menge von Eingaben in eine endliche Menge von Ausgaben zu überführen und sind daher nicht an statische Vektorgrößen gebunden. [Don18]

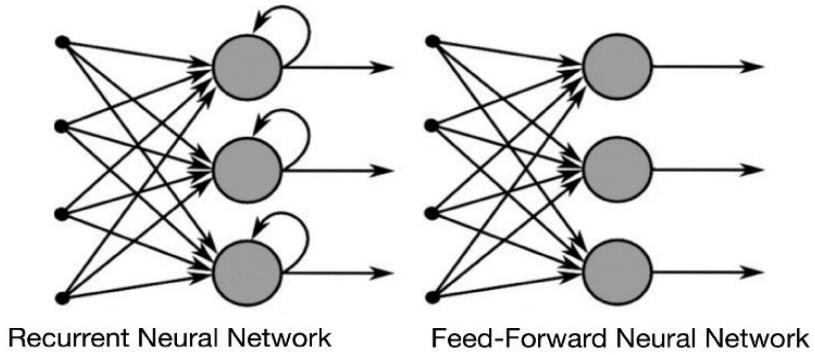



**Abbildung 4:** Vergleich zwischen RNN und feed-forward neural networks (Quelle: [Don18])

In Abb. 4 wird der unterschiedliche Informationsfluss in RNNs im Vergleich zu herkömmlichen Feed-Forward Netzen veranschaulicht. Ein RNN kann als eine vernetzte Sequenz von mehreren Kopien des neuronalen Netzes betrachtet werden, welche nacheinander mit der Backpropagation trainiert werden. Diese Betrachtungsweise eines RNNs bezeichnet man als Dekonvolution. In der Dekonvolution eines RNNs betrachtet man die einzelnen, miteinander verbundenen Zustände des Netzes über verschiedene Zeitschritte. Das RNN wird entfaltet und da es in jedem Zeitschritt einen anderen Zustand hat wird jeder Zeitschritt der Sequenz als Kopie des eigentlichen Netzes betrachtet. Ein Backpropagation Algorithmus, welcher in RNNs eingesetzt wird ist die sogenannte Backpropagation Through Time (BTT). Hierbei wird der Backpropagation Algorithmus für jede RNN Zelle, daher für jede Kopie des Netzes durchgeführt, da der Fehler eines Zeitschrittes von dem vorherigen Zeitschritt abhängig ist. Abb. 5 illustriert die Dekonvolution eines RNNs, in der $A$ ein neuronales Netz, $h_t$ die Netzausgabe eines Zeitschrittes und $X_t$ die Netzeingabe eines Zeitschrittes darstellt. [Don18]

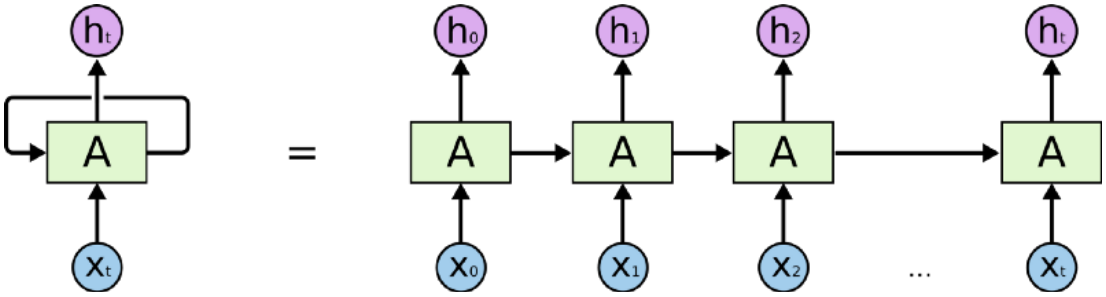



**Abbildung 5:** Dekonvolution eines RNNs (Quelle: [Don18])

Eine alternative Herangehensweise ist das Teacher Forcing. Dies ist eine alternative Strategie für die Backpropagation in RNNs. Hierbei werden Vorhersagen einzelner Zeitschritte zu einem gewissen Anteil anhand der Zielvorgaben korrigiert, statt das System mit einem hohen Fehlergradienten zu strafen. Diese Technik kann den Trainingsprozess von RNNs deutlich beschleunigen. [Bro17]

Beim Verarbeiten großer Sequenzen mit herkömmlichen RNNs stößt man jedoch schnell auf zwei schwerwiegende Probleme, die das Lernverhalten negativ beeinflussen. Bei der Gradienten Berechnung während der Backpropagation, durch die das Ausmaß der Anpassungen an den Gewichten ermittelt wird, kommt es beim Lernen von Beziehungen über größere Distanzen häufig zu Problemen, welche als verschwindende oder explodierende Gradienten bekannt sind. Explodierende Gradienten bezeichnen den Umstand, dass der Lernalgorithmus fehlerhafte Gradienten berechnet, was zu einer Überreaktion bei der Anpassung der Gewichte führt. Dieses Fehlverhalten führt durch das Multiplizieren der Gradienten über die verschiedenen Netzschichten zu einem exponentiellen Wachstum der Gewichte. Explodierende Gradienten können in instabilen Netzen resultieren, was sich wiederum äußerst negativ auf das Training auswirkt. [Gér17]

Bei dem Problem der verschwindenden Gradienten handelt es sich um ein vergleichbares Fehlverhalten. Hierbei werden die Gradienten während der Backpropagation stetig kleiner, was dazu führt, dass die Gewichte in den untersten Schichten zwar angepasst werden, dieser Einfluss jedoch mit jeder weiteren Schicht schwächer wird und darin resultiert, dass die Gewichtsanpassungen in den oberen Schichten so gering sind, dass die Gewichte dieser Schichten beinahe unverändert bleiben. Das Ergebnis ist ein starkes Ungleichgewicht im Lernfortschritt der einzelnen Schichten, welches verhindert, dass bestimmte Verbindungen zwischen den Daten einer Sequenz vom Netz erlernt werden können. Diese Probleme treten vor allem dadurch auf, dass größere Sequenzen in RNNs mit der Anzahl der Zeitschritte korrelieren und die Abhängigkeiten zwischen den Netzeingaben weit entfernter Zeitschritte für das System schwierig zu identifizieren sind. [Bro18]

Um diesem Problem entgegenzuwirken wurden verschiedene Lösungsansätze entwickelt, um herkömmliche RNNs so zu erweitern, dass die Informationen der Abhängigkeiten über mehrere Zeitschritte hinweg erhalten bleiben. Im Zuge des praktischen Teils dieser Arbeit werden die verwendeten RNNs zunächst durch sogenannte Gated Recurrent Units (GRU) erweitert. Diese verändern den Informationsfluss eines RNNs mit Hilfe von zwei speziellen Vektoren, welche Update-Gate und Reset-Gate genannt werden. Diese Vektoren können darauf trainiert werden, bestimmte Informationen zu speichern und diese vor den Einflüssen von verschwindenden und explodierenden Gradienten abzuschirmen. Dabei lernen sie zu entscheiden welche Informationen wichtig sind und das Netz unverändert passieren sollen und welche für die Vorhersage eine untergeordnete Rolle spielen und verworfen werden können. Der Update-Gate Vektor hilft dem Modell zu bestimmten, welche der bereits verarbeiteten Informationen aus vorherigen Zeitschritten für zukünftige Zeitschritte erhalten bleiben sollen. Analog dazu hilft der Reset-Gate Vektor dem Modell zu bestimmen welche Informationen verworfen werden können. [Kos17]

Ein weiterer bewährter Lösungsansatz, um Langzeitbeziehungen über mehrere Zeitschritte zu erhalten und den genannten Problemen entgegenzuwirken sind sogenannte LSTMs (Long Short-Term Memory). Hierbei handelt es sich um eine spezielle Form von RNNs, welche eigens zu dem Zweck entwickelt wurden, um Langzeitbeziehungen während des Trainings zu erhalten. LSTM Zellen sind ähnlich aufgebaut wie GRU Zellen. Der wesentliche Unterschied zwischen GRU und LSTM liegt darin, dass LSTM Zellen über einen speziellen Zellenstatus verfügen, mit dessen Hilfe diese Informationen gezielt weitergeben können. GRU Zellen geben diese Informationen direkt über die Hidden-States weiter. LSTM Zellen verfügen neben einem Output-Gate auch über ein Input - und ein Forget-Gate, während GRU die Funktionalität dieser beiden Tore auf das Reset-Gate reduziert. [Ngu18]

## 2.3 Seq2seq Framework

Das sequence to sequence learning (Seq2Seq) wurde Ende 2014 von Google vorgestellt. Hierbei handelt es sich um ein Framework, welches eine Encoder-Decoder Architektur verwendet. Das Seq2Seq Framework stellt einen Ende-zu-Ende Lösungsansatz für das sequenzielle Lernen dar. Die Encoder-Decoder Architektur wird im vorgestellten Paper mit zwei mehrschichtigen LSTM Netzen realisiert. [Sut14]

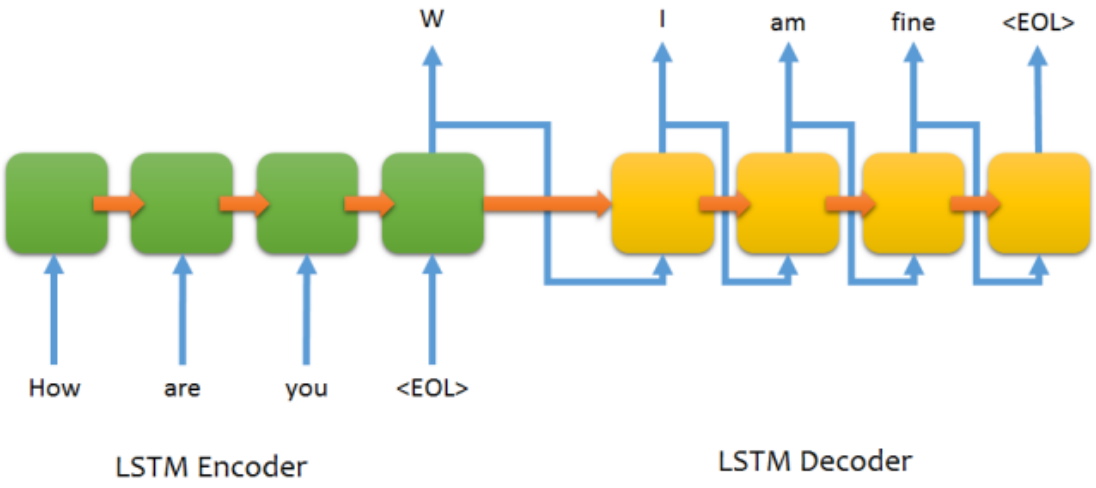



**Abbildung 6**: Abstraktes Beispiel der Encoder-Decoder-Architektur (Quelle: [Com16])

Der Encoder erhält eine Sequenz als Eingabe und liest in jedem Zeitschritt ein Sequenztoken ein und bildet dieses auf einen Vektor fester Größe ab. Dieser Vektor wird auch Kontext- oder Gedankenvektor genannt, da er die numerische Repräsentation der Bedeutung der abgebildeten Sequenz darstellt. Das LSTM des Decoders erhält diesen Vektor als Eingabe. Dieser extrahiert die Kontextinformationen aus dem Vektor in jedem Zeitschritt und generiert auf Grundlage dessen Schritt für Schritt eine Ausgabesequenz. Die LSTM Netze, welche für Encoder und Decoder eingesetzt werden, haben die nützliche Eigenschaft, dass sie lernen können Eingabesequenzen beliebiger Größe auf einen Vektor fester Größe abzubilden. Dabei korreliert die Distanz der einzelnen Kontextvektoren untereinander mit der Ähnlichkeit des zugrundeliegenden Satzkontextes. [Com16]

Die Vektorrepräsentationen für Paraphrasen haben beispielsweise eine geringe Distanz zueinander, während Sätze deren Bedeutungen sich stark voneinander unterscheiden entsprechend weit voneinander entfernt sind. Da jede Eingabesequenz unabhängig von ihrer Länge auf einen Vektor fester Größe abgebildet wird, welcher die Bedeutung des Satzes repräsentieren soll, stellt dieser Vektor einen Flaschenhals in der Verarbeitung dar. Ein wichtiges Verarbeitungsmerkmal des Seq2Seq Frameworks ist, dass es die Eingabe-sequenzen in umgekehrter Reihenfolge auf Vektoren abgebildet, um diesem Problem entgegenzuwirken. Dies soll das Optimierungsproblem minimieren und dabei helfen Langzeitabhängigkeiten aufzudecken, um das Verarbeiten langer Sequenzen zu ermöglichen. [Sut14]

Ein anderer, sehr bewährter Lösungsansatz, um die Performanz in der Encoder-Decoder Architektur zu optimieren und dem Flaschenhals, welcher aus der fixen Größe des Kontextvektors resultiert entgegenzuwirken ist der sogenannte Attention Algorithmus.

Hierbei bildet der Encoder, anders als in der herkömmlichen Encoder-Decoder-Architektur die Eingabesequenz nicht auf einen einzigen Kontextvektor ab, sondern auf eine Sequenz solcher Vektoren. Außerdem wird dem Netz des Decoders ein Attention-Layer hinzugefügt, welcher dazu dient automatisiert die einzelne Kontextvektoren aus der Vektorsequenz auszuwählen, welche für die Vorhersage des nächsten Wortes in der Ausgabesequenz notwendig sind. Somit wird die Abhängigkeit zwischen der Performanz des Systems und der Länge der Eingabesequenzen deutlich verringert. Damit der Decoder weiß, welcher Teil der Ausgabesequenz im aktuellen Zeitschritt zu beachten ist, werden sogenannte Ausrichtungsmodelle verwendet. Das Ausrichtungsmodell ist auch als Score-Funktion bekannt und berechnet nicht normalisierte Wahrscheinlichkeiten, welche Energien genannt werden. Aus den Energien des jeweiligen Zustands des Decoders und des Encoders kann die Ausrichtung und Übereinstimmung der beiden Zustände berechnet werden. [Bah16]

## 2.4 Word-Embeddings

Word-Embeddings werden in Systemen zur Verarbeitung natürlicher Sprache eingesetzt und erlauben das Erlernen einer Vektorraumrepräsentation von Wörtern und offenbaren semantische und syntaktische Relationen zwischen diesen. Word-Embeddings entstehen häufig als Nebenprodukt, während des Trainings eines Feed-Forward Netzes, welches Wörter als Eingabe erhält und diese als Vektoren interpretiert. Durch die Anpassung der Gewichte während der Backpropagation ergeben die Gewichte des ersten Layers die Word-Embeddings für die erlernten Wörter. Heute werden vermehrt Methoden wie Glove und Word2vec verwendet, da diese das explizite Ziel verfolgen Word-Embeddings zu erzeugen und dahingegen optimiert wurden allgemeine, statt aufgabenspezifische semantische Beziehungen zu kodieren. [Rud16]

Mit Hilfe der trainierten Word-Embeddings ist es möglich verwandte Wörter für ein bestimmtes Wort zu ermitteln, welche im gleichen Kontext eingesetzt werden können. Dies ist besonders hilfreich für die Paraphrasierung von Texten, da hierbei Alternativen für einen bestimmten Text angestrebt werden, wobei die zugrundeliegende Aussage nicht verändert werden darf. [Kar18]

### 2.4.1 **Word2Vec**

Word2Vec wurde 2013 von Google entwickelt und stellt zurzeit eine der meist verbreiteten Methoden zum Erlernen von Word-Embeddings dar. Bei dieser Methode werden flache neuronale Netze eingesetzt, welche als Eingabewert einen One-Hot-Enkodierungsvektor eines Wortes erhalten und basierend auf dessen Kontext versuchen ein Ziel-Wort vorherzusagen. Dabei ermittelt das Netz den Fehler zwischen Eingabe- und Ziel-Wort. Das Netz erlernt während des Trainings die Vektorrepräsentation des Ziel-Wortes, um den Fehler zu minimieren. [Kar18]

# 3 Problemstellung

In diesem Kapitel werden im Zuge der Analyse die Anforderungen ermittelt aus denen anschließend die Spezifikation abgeleitet wird.

## 3.1 Analyse

Der Analyseteil konzentriert sich auf die Erhebung der Anforderungen, welche die Grundlage für die Spezifikation darstellen.

### 3.1.1 Einsatzzweck des Systems

Das System soll als Ergänzung für NLG Prozesse eingesetzt werden, um die Komplexität und die Verschachtelung von NLG Templates erheblich zu reduzieren. Die eingesetzten NLG Prozesse des Unternehmens LangTec basieren zumeist auf Jinja2 Templates. Diese Templates werden von Linguisten und Softwareentwicklern entwickelt. Die Entwicklung und die Wartung solcher Templates sind in der Regel sehr aufwändig, da deren Komplexität je nach Verschachtelungs- und Verzweigungsgrad sehr hoch ist. Ziel des Paraphrasing-Systems ist es den Entwicklungs- und Administrationsaufwand für solche Systeme zu verringern. Der angestrebte Workflow soll so aussehen, dass ein NLG Template einen Text erzeugt und dieser im Anschluss durch ein Modell, welches auf einer einschlägigen Domäne trainiert wurde paraphrasiert wird. Dabei werden Datenfelder wie z.B. Ort-, Temperatur- oder Prozentwerte durch Platzhalter ersetzt. Das Ergebnis des Prozesses ist eine alternative Textvariante, welche den gleichen Informationsgehalt wie der ursprüngliche Text aufweist, jedoch in Bezug auf diesen über eine möglichst hohe Textvarianz verfügt. Sowohl die Input-Texte als auch die erzeugten Paraphrasen sollten die gleiche Formatierung und Enkodierung verwenden. Das System sollte unabhängig von anderen Systemen und Netzwerkeinflüssen arbeiten, um die Ausfallsicherheit und Modularität zu erhöhen.

### 3.1.2 Ergebnisqualität

Das Ergebnis des Systems sollen trainierte Modelle sein, welche als auslieferbares Produkt dazu dienen, Paraphrasen für Eingabetexte einer Domäne zu erzeugen. Diese alternativen Textinhalte sollen die ursprüngliche Aussage der Eingabetexte nicht verändern. Die

Paraphrasen sollten sich jedoch auf sprachlicher Ebene möglichst stark von den Eingaben unterscheiden. Das heißt, dass die Textvarianz zwischen dem Eingabetext und der Paraphrase sollte möglichst hoch sein. Die Textvarianz ist in Bezug auf den Einsatzzweck des Systems ein wichtiges Qualitätsmerkmal, da die Texte beispielsweise an unterschiedliche Kunden ausgeliefert werden und auf unterschiedlichen Plattformen publiziert werden sollen. Die generierten Texte sollen über ein möglichst hohes sprachliches Niveau verfügen und frei von Rechtschreib- und Grammatikfehlern sein.

### 3.1.3 **Domänenauswahl**

Die Domäne welche für den konzeptionellen Beweis im Zuge dieser Bachelorarbeit genutzt wird sollte den angestrebten Workflow demonstrieren, ohne dass Umfang und Komplexität der Domäne den Rahmen dieser Arbeit sprengen. Die Kopplung zwischen dem System und der ausgewählten Domäne sollte so gering wie möglich sein, um zu gewährleisten, dass das System auf beliebigen Domänen trainiert werden kann, ohne dass es selbst adaptiert werden muss.

### 3.1.4 **Systemschnittstellen**

Die Daten-Schnittstellen auf deren Grundlage die Eingabetexte von NLG Templates erzeugt werden, bringen aufgrund der notwendigen Aktualität der Texte meist zeitkritische Anforderungen mit sich. Es ist es wichtig, dass der Generierungsprozess maximal eine Stunde in Ansprung nimmt, da die Daten und Eingabetexte teilweise stündlich aktualisiert werden. Die vom System für die Generierung benötigten Ressourcen sollen sich auf die trainierten Modelle, eine auf Basis der Trainingsdaten erstellten Sprache und die verwendeten Eingabetexte beschränken. Das System soll über einen Evaluationsprozess verfügen, dessen Methoden und Funktionen von NLG Systemen verwendet werden können, um die Paraphrasierung in deren Workflow einzubinden.

## 3.2 Spezifikation

Dieses Unterkapitel enthält die Spezifikation des Paraphrasing-Systems basierend auf den Anforderungen, welche aus der vorausgegangenen Analyse hervorgehen. Die Spezifikation unterteilt sich in die funktionalen und nicht funktionalen Anforderungen, welche an das System gestellt werden.

### 3.2.1 Funktionale Anforderungen

Im Folgenden werden basierend auf der Analyse die funktionalen Anforderungen erhoben, welche an das zu entwickelnde Paraphrasing-System gestellt werden. Die linke Spalte benennt die entsprechende Anforderung, während die rechte Spalte eine Beschreibung dieser enthält.

| Nr. | Funktionale Anforderung | Beschreibung |
|---|---|---|
| 1 | Das Paraphrasing-System soll die Komplexität, sowie den Administrations- und Pflegaufwand bestehender NLG-Templates reduzieren können. | Das System soll als Ergänzung für NLG Templates eingesetzt werden können. Diese Templates verfügen oftmals über komplexe Verzweigungen, um alternative Texte zu generieren. Durch die Einbettung des Paraphrasing-Systems, soll die Komplexität reduziert werden, da die Aufgabe der Generierung von Alternativtexten in das Paraphrasing-System ausgelagert werden soll. |
| 2 | Der BLEU Score soll im Vergleich von Originaltext und Paraphrase mindestens 0.5 betragen, um sicherzustellen, dass die zugrunde liegende Aussage nicht verfälscht wurde. | Die ursprüngliche Aussage des Textes, soll nicht verändert werden. Als Qualitätsmaßstab wird hierbei der BLEU-Score verwendet. Es wurde basierend auf Vergleichen mit NMT-Modellen entschieden, dass der BLEU-Score für das Paraphrasing-System mindestens 0.5 betragen soll. Ein Score von 0.5 und höher gilt im Bereich der neuronalen Übersetzungsprogramme als optimal. [Isi18]<br><br>Andere Paraphrasing-Systeme erreichten im Zusammenspiel mit menschlichen Bewertungen der Paraphrasen ähnliche Ergebnisse. [Bar13] |

| | | |
|---|---|---|
| 3 | Die Kosinus-Ähnlichkeit soll im Vergleich von Originaltext und Paraphrase höchstens 0.7 betragen. | Als Qualitätsmaß für die Textvarianz zwischen Originaltext und Paraphrase wird die Kosinus-Ähnlichkeit verwendet. Da die Textvarianz ein entscheidendes Qualitätsmerkmal für die generierten Texte darstellt, soll die Kosinus-Ähnlichkeit maximal 0.7 betragen. [Cro15] |
| 4 | Die generierten Texte sollen sprachlich korrekt sein. | Die resultierenden Paraphrasen sollen sowohl in Bezug auf die Rechtschreibung als auch bezüglich der Grammatik fehlerfrei sein, um den Aufwand der Nachbearbeitung zu reduzieren. |
| 5 | Das System soll Paraphrasing-Modelle trainieren und speichern. | Das Ergebnis des Paraphrasing-Systems sollen trainierte Modelle bestimmter Domänen sein, welche für die Generierung beliebig vieler Paraphrasen verwendet werden sollen. |
| 6 | Das System soll in der Produktivumgebung einen in UTF-8 formatierten Text als Eingabe erhalten und eine Paraphrase als Ausgabe liefern. | Das System soll die Verarbeitung von Texten im UTF-8 Format unterstützen und auf Basis der trainierten Modelle für jeden Eingabetext je eine Paraphrase als Text im UTF-8 Format zurückgeben. |

### 3.2.2 Nicht funktionale Anforderungen

In diesem Abschnitt werden die nicht funktionalen Anforderungen, welche an das Paraphrasing-System gestellt werden, erhoben. Die linke Spalte enthält eine Beschreibung der Anforderung, während die rechte Spalte die jeweilige Kategorie nach ISO9126 enthält, in die diese eingeordnet wird.

| Nr. | Nicht funktionale Anforderung | Kategorie nach ISO9126 |
|---|---|---|
| 1 | Die Generierung der paraphrasierten Wettertexte darf nicht länger als eine Stunde dauern, da die Wetterdaten stündlich aktualisiert werden. Im besten Fall soll die Generierung wenige Sekunden in Anspruch nehmen. | Effizienz |

| | | |
|---|---|---|
| 2 | Die einzigen Ressourcen sollen das trainierte Modell und die verwendete Sprache sein, welches jederzeit ausgetauscht werden kann. | Effizienz |
| 3 | Die Generierung von Paraphrasen durch ein trainiertes Modell benötigt keine bestehende Internetverbindung | Zuverlässigkeit |
| 4 | Das System soll so konzipiert sein, dass es für beliebige Domänen eingesetzt werden kann. Daher soll die Auswahl der Domäne völlig unabhängig vom eigentlichen System sein. | Übertragbarkeit |
| 5 | Das Produkt sollte ein in sich geschlossenes System sein, damit es später möglich ist dieses als Blackbox in NLG-Prozesse bestehender Systeme einzubetten. | Übertragbarkeit |
| 6 | Das System soll den Benutzer darüber informieren, wenn die Textvarianz einer generierten Paraphrase zu gering ist. | Benutzbarkeit |

# 4 Konzept

Im vorherigen Kapitel wurden die Anforderungen spezifiziert, welche an das zu entwickelnde Paraphrasing-System gestellt werden. Darauf aufbauend werden in diesem Kapitel unter Einsatz von wissenschaftlichen Methoden des Software-Engineerings der konzeptionelle Aufbau und das Design des Systems entwickelt.

## 4.1 Architektur

Zu Beginn der Konzeption soll die Architektur des Paraphrasing-Systems festgelegt werden. Hierfür wird das System in verschiedene Komponenten unterteilt und in seiner Gesamtheit aus verschiedenen Blickpunkten betrachtet. Zum einen aus der Sicht des Benutzers, dessen Bestreben es ist domänenspezifischen Paraphrasen mit Hilfe von bereits trainierten Modellen zu erzeugen, welche das resultierende Produkt des Paraphrasing-Systems darstellen. Außerdem kann das System aus der Entwicklersicht betrachtet werden, in welcher der Fokus auf dem Training der Modelle anhand von Trainingsdaten liegt. Hierbei stellt auch die Encoder-Decoder-Architektur des Seq2Seq-Frameworks einen wichtigen Bestandteil des Systems dar. Dieses Kapitel beschreibt die Komponentenarchitektur der beiden genannten Sichtweisen auf das Gesamtsystem. Das System kann aus verschiedenen Standpunkten betrachtet werden. Abb. 7 zeigt die Komponentenarchitektur aus der Sicht des Endverbrauchers. Hierbei werden die Eingabetexte für eine bestimmte Domäne von einem NLG-System des Benutzers erzeugt. Dieser Originaltext wird anschließend an das Paraphrasing-System übergeben. Auf Basis der trainierten Gewichte des Modells wird der Eingabetext satzweise verarbeitet und paraphrasiert und anschließend zu einem Text im UTF-8 Format zusammengeführt. Zum Schluss wird das Endergebnis an das NLG-System des Benutzers übermittelt.

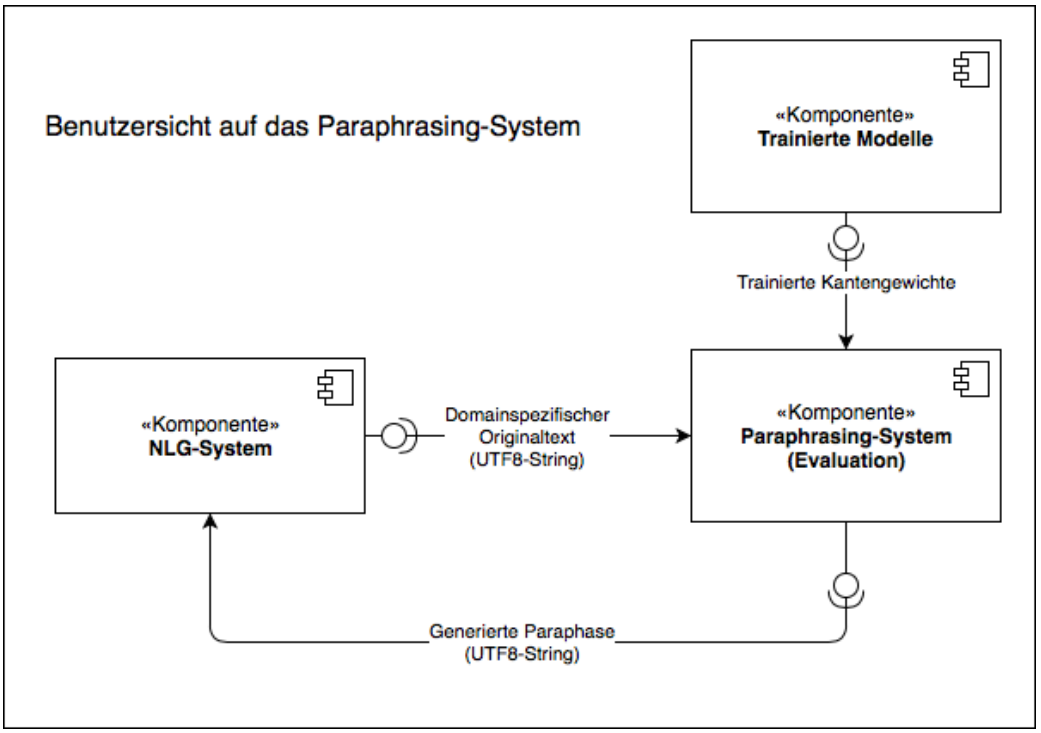



**Abbildung 7**: Benutzersicht auf das System

Abb. 8 zeigt die Komponentenarchitektur aus der Sicht der Entwicklung. Hierbei liegt der Fokus auf dem Trainingsprozess aus dem die angestrebten Models für domänenspezifischen Paraphrasen hervorgehen. Hierbei werden die benötigten Daten von einer Schnittstelle bereitgestellt, welche anschließend von einem NLG-System genutzt werden, um auf Basis dessen unterschiedliche Texte für denselben Dateninput zu generieren. Das System liest die resultierenden Rohtexte ein, bereinigt diese und speichert sie als Paraphrasen-Tupel ab. Auf Grundlage dieser Trainingsdaten wird eine Sprache erzeugt und in einem Model gespeichert. Während des Trainings werden die Paraphrasen-Tupel in vektorisierter Form gemeinsam mit der erzeugten Sprache an den Encoder übergeben. Das Ergebnis aus der Verarbeitung im Encoder ist ein Kontextvektor, welcher wiederum als Eingabe für die Decoder dient. Dieser erzeugt auf Basis des Vektors eine Ausgabesequenz, welche mit dem Ziel-Label verglichen wird. Im Anschluss werden die Gewichte der Netze optimiert. Dieser Vorgang wiederholt sich in jeder Epoche des Trainings. Nach Abschluss des Trainings werden die resultierenden Netze als Modelle gespeichert und können in der Evaluation für die Vorhersage von Paraphrasen genutzt werden.

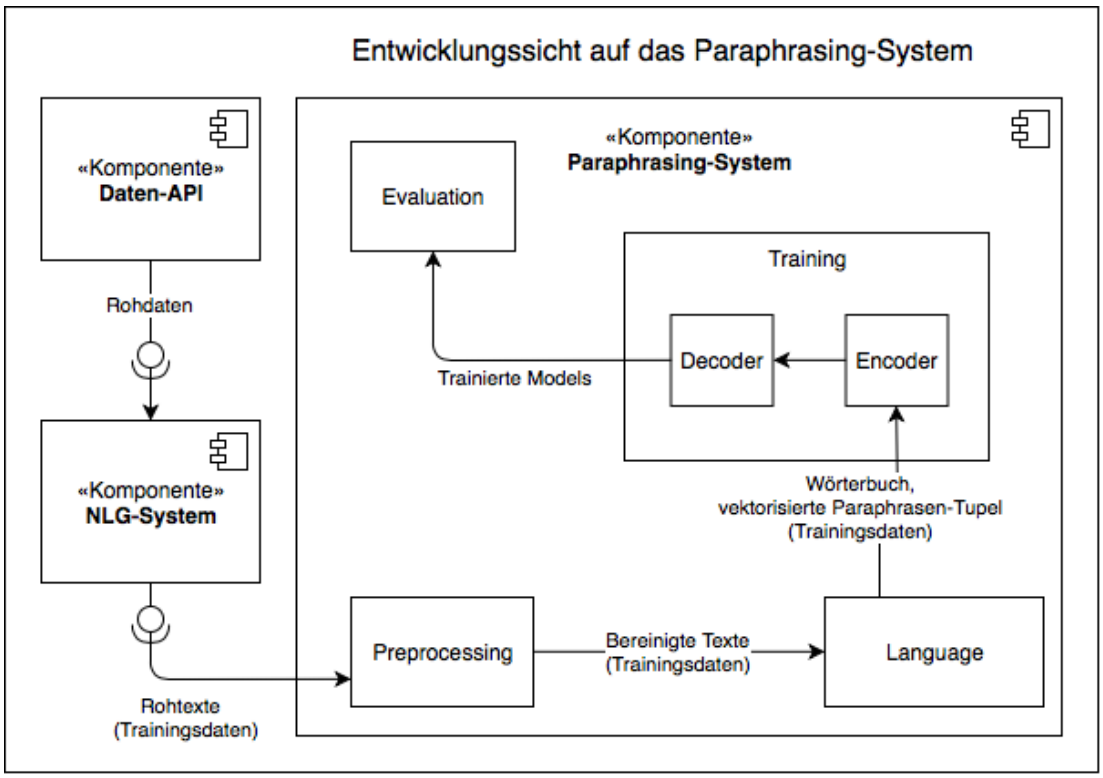



**Abbildung 8**: Entwicklersicht auf das System

## 4.2 Entwurf

In diesem Kapitel werden im Zuge des konzeptionellen Designs die Entwurfsentscheidungen getroffen. Diese stellen unter Berücksichtigung der spezifizierten Anforderungen die Grundlage für die Realisierung des Systems dar.

### 4.2.1 Entwurfsentscheidungen über Entwicklungswerkzeuge

Als Programmiersprache für die Implementierung des angestrebten Paraphrasing-Systems wurde Python 3.6 ausgewählt, da diese Sprache aufgrund ihrer übersichtlichen Struktur und den zur Verfügung stehenden Programmbibliotheken ein ideales und beliebtes Werkzeug für die Entwicklung von Systemen im Bereich des maschinellen Lernens darstellt. Darauf aufbauend wurde PyTorch als Programmbibliothek für die Umsetzung der Deep-Learning-Prozesse eingesetzt. PyTorch bietet einen großen Umfang an Deep-Learning-Funktionen und

ist aufgrund des hohen Abstraktionsgrades in der Implementierung und der übersichtlichen Framework-Dokumentation unkompliziert zu erlernen.

Darüber hinaus unterstützt PyTorch verstärkt die Berechnung über GPU, welches das Training des Paraphrasing-Systems auf großen Datenmengen erleichtert. Die verwendeten Algorithmen benötigen in vielen Fällen deutlich weniger Rechenoperationen als vergleichbare Frameworks, wie z.B. TensorFlow was zu Gunsten der Geschwindigkeit des Trainings geht. Dies ist besonders bei Performanz sensitiven Systemen wie dem zu entwickelnden Paraphrasing-System von Vorteil, da die verwendeten Daten oft nur für einen kurzen Zeitraum gültig sind und es daher notwendig ist, die benötigten Modelle in einem möglichst kleinen Zeitfenster zu erzeugen. Als Entwicklungsumgebung wurde PyCharm verwendet. Ein Vorteil dieser IDE ist, dass sie eine virtuelle Umgebung für die Entwicklung von Systemen bereitstellt. PyCharm unterstützt mit PEP8 Markierungen die Entwicklung in der formatierungssensitiven Sprache Python. Darüber hinaus verfügt die IDE über eine integrierte Versionskontrolle, eine Pythonkonsole und ein benutzerfreundliches Dateimanagement.

### 4.2.2 Entwurfsentscheidungen über die Domänenauswahl

Als exemplarisches Experiment soll das System auf der Grundlage von Wetterberichten trainiert werden. Die tägliche Nachfrage nach individuellen Wettertexten ist hoch, da es sich hierbei um Informationen handelt, welche sich kontinuierlich verändern und daher täglich, oder sogar mehrmals täglich aktualisiert werden müssen. Diese Domäne ist hinreichend komplex, um ein solches Experiment durchzuführen und ist dabei nicht so umfangreich, als dass sie den Rahmen dieser Arbeit sprengen würde. Die gewählte Domäne dient als konzeptioneller Beweis und beschränkt sich zunächst auf einzelne Wetterphänomene. Die Durchführung des Experiments zur Paraphrasierung ausgewählter Wettertexte soll zeigen, dass das System auf Texten einer beliebigen Domäne trainiert und evaluiert werden kann. Daher ist es vorgesehen, dass die Domänenauswahl streng vom eigentlichen System abgekapselt wird und die verwendeten Trainingsdaten die einzige Schnittstelle zwischen Domäne und System darstellen. Die Trainingsdaten für das Paraphrasing-System stammen von einem NLG-System des Unternehmens LangTec aus Hamburg. Das System verwendet Jinja2-Templates, welche über komplexe Verzweigungen und eine Vielzahl von Alternativen verfügen und mit Hilfe von Generatoren Texte für verschiedene Domänen bereitstellen.

Die Daten der Wettervorhersagen werden vom Deutschen Wetterdienst (DWD) bereitgestellt und während des Generierungsprozesses heruntergeladen. Der Generator analysiert diese Daten und bildet sie in Dataframes ab, um mit Hilfe der von Linguisten erstellten Templatestrukturen möglichst hochwertige Texte zu erzeugen. Unter Verwendung einer lokalen Kopie des Generators für Wettertexte wurden basierend auf den Daten des DWD mehrere Texte für jeden Datensatz generiert und die daraus resultierenden

Vorhersagen als Paare von Paraphrasen abgespeichert. Dynamische Daten in den Trainingsdaten wie beispielsweise einzelne Temperaturwerte oder die Namen von Städten wurden durch Tokens ersetzt, um das Training des Paraphrasing-Systems zu vereinfachen.

Die Daten werden stündlich aktualisiert und sind online verfügbar unter [Src1]

Darauf aufbauend sollen die akquirierten Trainingsdaten im Zuge eines Experiments zur Augmentierung der Daten manuell erweitert werden. Hierfür wird untersucht, wie groß der Einfluss einzelner Paraphrasen-Tupel in den Trainingsdaten auf das Ergebnis ist. Außerdem wird untersucht, wie sich die manuelle Erweiterung der Trainingsdaten auf die Textqualität der Paraphrasen auswirkt. Die Daten werden in einen Trainings- und einen Testdatensatz aufgeteilt, wobei 80% der vorhandenen Daten für das Training und 20% für die Evaluation des Systems eingesetzt werden.

### 4.2.3 **Technische Entwurfsentscheidungen**

Um das Paraphrasing-System zu realisieren, werden im Zuge des praktischen Teils dieser Bachelorarbeit verschiedene Konzepte und Methoden aus dem Bereich der künstlichen Intelligenz benötigt. Für die sequenzielle Textverarbeitung von Paraphrasenpaaren wird das Sequence-2-Sequence Framework von Google eingesetzt. Es dient dazu Eingabesequenzen wortweise in Ausgabesequenzen zu überführen. Das Framework bedient sich der Encoder-Decoder-Architektur, um Paraphrasing-Modelle zu trainieren. Hierbei soll das System auf Basis der Trainingsdaten mit Hilfe eines künstlichen, neuronalen Netzes trainiert werden, sodass es in der Lage ist einschlägige Eingabetexte sinngemäß zu interpretieren und in numerischer Form in einem Kontextvektor abzubilden. Ein weiteres Netz soll darauf trainiert werden, die notwendigen Informationen aus diesem Kontextvektor zu extrahieren, anschließend zu verarbeiten und auf Grundlage dieser Informationen eine endliche Menge von Paraphrasen zu dem gegebenen Eingabetext zu generieren. Für die Netze des Encoders und Decoders werden GRU-Units eingesetzt, um dem Problem der verschwindenden Gradienten entgegenzuwirken. Ein wesentlicher Vorteil ist, dass GRU-Units häufig bereits auf geringen Mengen von Trainingsdaten gute Ergebnisse in der Verarbeitung natürlicher Sprache liefern. GRU-Units verfügen über weniger Tore und Rechenoperationen als LSTM-Units, daher können solche Netze, welche GRU-Units verwenden in der Regel schneller trainiert werden. In einem Experiment soll jedoch untersucht werden, wie sich der Austausch der GRU-Units durch LSTM-Units auf das Ergebnis auswirkt. LSTM-Units sind besser für lange Sequenzen geeignet und da die Trainingsdaten meist aus längeren Sätzen bestehen könnte sich dies positiv auf das Ergebnis auswirken. Um Eingabesequenzen unterschiedlicher Längen in Abhängigkeit von der fixen Länge des Kontextvektors des Encoders verarbeiten zu können wird ein Attention-Layer im Netz des Decoders eingesetzt. Als Word-Embeddings für die beiden Netze wurde ein vortrainiertes Word2Vec Model eingesetzt. Das Model ist online verfügbar unter [Src2]

## 4.3 Metriken

Um die Ergebnisse des entwickelten Paraphrasing-Systems hinsichtlich ihrer Korrektheit und Vielfältigkeit zu bewerten werden als Maßnahme der Qualitätssicherung zwei Metriken eingesetzt. Zu den verwendeten Metriken gehören der BLEU-Score und die Kosinus-Ähnlichkeit. Diese Metriken dienen als Maßstab, um die Ergebnisse der durchgeführten Experimente zu bewerten und in Relation zu setzen. Das Ziel ist es eine möglichst geringe Kosinus-Ähnlichkeit und einen möglichst hohen BLEU Score zu erreichen.

### 4.3.1 **BLEU**

Der Bilingual Evaluation Understudy (BLEU) Score dient dazu die Relation zwischen Systemeingabe und Systemausgabe in Bezug auf die Bedeutung des Satzes zu untersuchen. Der BLEU Score ist ein wichtiger Maßstab für die Korrektheit des Paraphrasing-Systems. Mit dieser Metrik kann überprüft werden, zu welchem Maß die Anforderung, dass der paraphrasierte Text die Bedeutung des ursprünglichen Textes nicht verändern darf erfüllt wurde. Der BLEU Score wurde ursprünglich entwickelt, um die Ergebnisse von maschinellen Übersetzungsprogrammen zu bewerten. Die Ergebnisse dieser Metrik liegen zwischen 0 und 1, wobei 1 das angestrebte Optimum ist. Der Algorithmus misst die Ähnlichkeit zweier Texte durch das Vergleichen ihrer N-Gramme und ist unabhängig von der Reihenfolge der Wörter. In der folgenden Formel bezeichnet $c$ die Länge der Ausgabe und $r$ die effektive Referenzkorpuslänge. [Pap02]

$$BP = \begin{cases} 1 & if\ c > r \\ e^{(1-r/c)} & if\ c \leq r \end{cases}$$

$$BLEU = BP \cdot \exp(\sum_{n=1}^{N} w_n \log\ p_n)$$

### 4.3.2 **Kosinus-Ähnlichkeit**

Da die Textvarianz jedoch ein wichtiges Qualitätsmerkmal der generierten Paraphrasen ist, müssen die Ergebnisse auch hinsichtlich ihrer Vielfältigkeit überprüft werden. Zu diesem Zweck wird ergänzend zum BLEU Score die sogenannte Kosinus-Ähnlichkeit als Metrik verwendet. Hierbei werden die zu vergleichenden Texte in Vektoren umgewandelt, indem beispielsweise das Vorkommen einzelner Worte gezählt wird, um den Kosinus zwischen diesen zu ermitteln. Mit dieser Metrik soll geprüft werden, wie ähnlich die zu vergleichenden

Texte hinsichtlich der verwendeten Wörter sind. Eine geringe Kosinus -Ähnlichkeit impliziert daher ein hohes Maß an Textvarianz. [Gup18]

$$\cos(\theta) = \frac{A \cdot B}{||A||||B||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

### 4.3.3 **Negative Log Likelihood Loss**

Als Fehlerfunktion wird während des Trainings NLLL (Negative Log Likelihood Loss) verwendet, um für jede Netzausgabe die Abweichung zwischen Vorhersage und Label zu berechnen. [Cha17]

$$L = -\frac{1}{n}\sum_{i=1}^{n} \log(y^{(i)})$$

NLLL wird als Metrik in der Evaluation der Ergebnisse verwendet, um die Approximation der Netzausgaben gegen die gegebenen Zielwerte während des Trainingsprozesses zu visualisieren. Sie dient ebenfalls nur als grobe Richtlinie für die Bewertung der Ergebnisqualität, da sprachliche und inhaltliche Qualität einer Paraphrase in Bezug auf einen Eingabesatz nicht von dieser Metrik erfasst werden kann.

# 5 Realisierung

Dieses Kapitel befasst sich mit der Umsetzung des zuvor entwickelten Konzepts und der konkreten Implementierung des daraus resultierenden Systems. Die Implementierung des Trainingsprozesses orientiert sich an der NMT-Applikation des Seq2Seq Beispiels von PyTorch. Die Implementierung des exemplarischen Systems zum Übersetzen von natürlichen Sprachen mit Hilfe von neuronalen Netzen von PyTorch ist online verfügbar unter [Src3]

## 5.1 Wörterbuch

Die Language-Klasse dient dazu ein indiziertes Wörterbuch auf Basis der in den Trainingsdaten enthaltenen Wörtern anzulegen. Jedes eingelesene Wort, welches sich noch nicht im Wörterbuch befindet wird diesem zusammen mit einem Index in einer Key-Value-Datenstruktur hinzugefügt. Dies dient dazu, die übergebenen Texte so aufzubereiten, dass diese als Eingabe und Zielwerte für die Netzschichten des Encoders und Decoders verwendet werden können. Zu diesem Zweck wird die Funktion variable_from_sentence(lang, sentence) eingesetzt, um den übergebenen Satz anhand der Indizes der einzelnen Wörter in einen Tensor abzubilden. Der resultierende Tensor kann dann während der Evaluation als Netzeingabe verwendet werden. Die Methode variables_from_pair(lang, pair) bildet zusätzlich die gewünschte Zielparaphrase für einen Satz auf einen Tensor ab, um diese während des Trainingsprozesses als Zielwert der Netzeingabe zu verwenden.

## 5.2 Datenaufbereitung

Die gesammelten Rohdaten aus den Templates, welche als Fließtexte im HTML-Format abgespeichert wurden, werden zunächst bereinigt, in UTF-8 kodiert und in einzelne Sätze aufgespalten. Für das Experiment der Datenaugmentierung werden bei Bedarf zusätzlich die manuell erzeugten Texte zur Erweiterung der Datenbasis geladen. Anschließend werden die Sätze in Trainings- und Testdaten aufgeteilt. Um Duplikate zu vermeiden und die eingelesene Reihenfolge zu annullieren werden die Daten in Set-Datenstrukturen abgespeichert. Die Sätze aus den Trainingsdaten, welche die gleichen Wetterdaten beschreiben, werden zu Paraphrasenpaaren zusammengefügt. Zuvor wird überprüft, ob:

- der jeweilige Satz die maximale Satzlänge aus der Systemkonfiguration nicht überschreitet
- die beiden Sätze nicht identisch sind
- keiner der Sätze in den Testdaten vorkommt
- der BLEU-Score der Sätze einen bestimmten Grenzwert erreicht, um Fehlern in der Semantik vorzubeugen

Zum Schluss werden die gesammelten Paraphrasenpaare aus der Set-Datenstruktur, zeilenweise in einer Textdatei gespeichert.

## 5.3 Train-Test-Split

Die gesammelten Daten werden während der Aufbereitung in einen Trainings- und einen Testdatensatz aufgeteilt. Dabei bestehen die Trainingsdaten aus Paraphrasenpaaren, daher aus einem Eingabesatz und einer Zielparaphrase. Die Testdaten für die Evaluation bestehen wiederum aus einzelnen Sätzen. Der Grund dafür liegt darin, dass die Testdaten für das System völlig unbekannt sein sollen und die Paraphrasenpaare welche dasselbe Phänomen beschreiben für unterschiedliche Eingaben dieselbe Paraphrase als Zielwert verwenden können. Die zugrundeliegenden Daten sind zunächst einzelne Sätze die jeweils mit allen Sätzen zu Paraphrasenpaaren kombiniert werden, welche dieselben Daten beschreiben. Während des Trainingsprozesses werden Paare benötigen, um daraus die Eingabe und den Zielwert der neuronalen Netze zu bestimmen. Daher werden 20% der Sätze in den Daten bereits vor dem Erstellen der Paraphrasenpaare entfernt und als Testdatensatz für die Evaluation gespeichert. Die verbleibenden 80% der Sätze werden zu Paaren kombiniert, welche anschließend als Trainingsdatensatz verwendet werden.

## 5.4 Aufbau des Encoders und Decoders

Der Encoder und der Decoder verwenden die Klasse PyTorch.nn.Module als Superklasse. Dabei handelt es sich um die Basisklasse für neuronale Netze in PyTorch. Beide Netze initialisieren ihre Embeddings mit den zuvor erwähnten, vortrainierten Word2Vec Embeddings mit Hilfe der Gensim-Bibliothek. Der Encoder bekommt die Anzahl der Wörter der erstellen Sprache aus dem Language-Objekt Anzahl der erwarteten Eingabefeature übergeben. Die Anzahl der Netzschichten und die Anzahl der Features für die Hidden-States werden über eine Konfigurationsdatei, im YAML-Format definiert. Auf Basis dieser Hyperparameter wird eine GRU-Unit als RNN-Zelle erstellt. Während der Propagierung werden die Größe des Vokabulars und die Größe des jeweiligen Embedding-Vektors des Tensors der Netzeingabe als Embeddings im jeweiligen Zeitschritt hinzugefügt. Anschließend wird die Form des Embedding-Tensors entsprechend der Länge der Eingabesequenz reshaped. Abb. 9 zeigt den Aufbau des Encoders als Module-Tree.

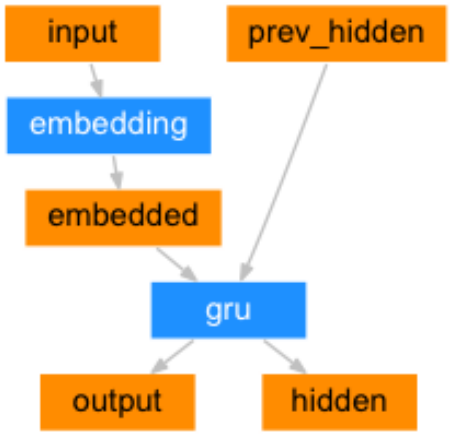



**Abbildung 9**: Aufbau des Encoders (Quelle: [Rob17])

Der Decoder bekommt ein Ausrichtungsmodell für den Attention-Layer, die Anzahl der Features für die Hidden-States, die Anzahl der Features der Netzausgabe, die Anzahl der Netzschichten und einen Dropout-Wert als Parameter übergeben. Als Aktivierungsfunktion für die Hidden-Layers wird die Relu-Funktion verwendet, während für den Output-Layer die Softmax-Funktion eingesetzt wird. Während der Propagierung wird auf Basis der Netzausgabe des Encoders ein Attention-Layer erstellt, dessen Gewichte verwendet werden um den Kontext des Encoder-Outputs zu interpretieren. Als Attention-Mechanismus wird eine Interpretation des von Luong et al. vorgestellten Ansatzes verwendet. [Luo15]

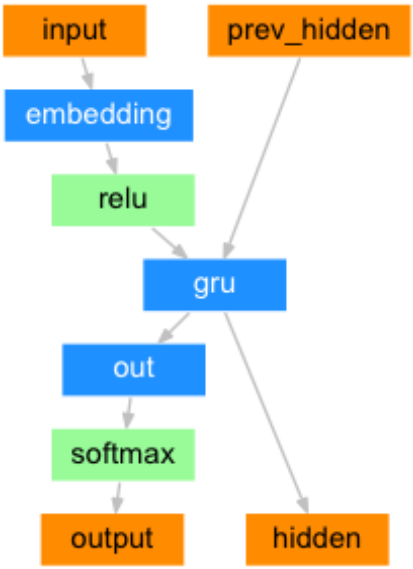



**Abbildung 10**: Aufbau des Decoders (Quelle: [Rob17])

## 5.5 Trainingsprozess

Zu Beginn des Trainingsprozesses wird mit Hilfe der Language-Klasse auf Basis der Trainingsdaten ein indiziertes Wörterbuch erstellt und im Pickle-Format gespeichert. Anschließend werden die Netze und deren Parameter über die Angaben in der Konfigurationsdatei initialisiert. In jeder Epoche des Trainings wird ein pseudozufälliges Paraphrasenpaar aus den Trainingsdaten ausgewählt. Der Eingabetext der Paraphrase wird dem Encoder als Netzeingabe übergeben, während die dazugehörige Paraphrase als Label verwendet wird, um den Wert der Fehlerfunktion für die Ausgabe des Decoders zu bestimmen. Als Fehlerfunktion wird NLLLoss() von PyTorch´s NN-Modul verwendet (negative log likelihood loss). Für die Backpropagation zur Anpassung der Gewichte beider Netze wird der Adam-Optimizer von PyTorch eingesetzt. Anschließend werden die auf Basis des übergebenen Paraphrasenpaars trainierten Netze als neue Instanzen zurückgegeben. Dadurch werden die Gewichte des Encoders und Decoders in jeder Epoche inkrementell optimiert. Anschließend wird ein erneut pseudozufällig gewähltes Paraphrasenpaar ausgewählt, welche mit den aktuellen Netzinstanzen evaluiert wird. Die Ergebnisse der Evaluation werden in der Konsole ausgegeben. Auf Basis der resultierenden Paraphrasen und des Ursprungstextes wird der BLEU-Score und die Kosinus-Ähnlichkeit berechnet. Zum Ende des Trainings werden die gesammelten Daten der Metriken in einem Graphen visualisiert und die trainierten Modelle der Netzinstanzen werden im Pickle-Format gespeichert.

## 5.6 Evaluation

Für die Evaluation der trainierten Modelle werden zunächst die gespeicherten Instanzen des Encoders, Decoders und des dazugehörigen Wörterbuches geladen. Das System erhält eine beliebige Eingabesequenz ohne dazugehörigen Zielwert als Eingabe und evaluiert diese mit Hilfe der geladenen Netzinstanzen. Als Ergebnis wird die resultierende Paraphrase, so wie die berechneten Metriken zurückgegeben. Während des Evaluationsworkflows werden alle Sätze der Testdatendatei zeilenweise eingelesen und Evaluiert. Als Ergebnis werden die Durchschnittswerte für die verwendeten Metriken ausgegeben.

# 6 Experimente und Auswertung

In diesem Kapitel werden die Ergebnisse dieser Arbeit vorgestellt. Das entwickelte System wird auf Basis verschiedener Baselines eingesetzt. Die daraus entstehenden Ergebnisse werden untersucht und bewertet. Zu Beginn wird die grundlegende Funktionalität des Paraphrasing-Systems geprüft. Anschließend werden im Zuge von Experimenten Veränderungen an der Baseline vorgenommen, um zu untersuchen, wie sich die Modifikationen auf die Qualität der Ergebnisse auswirken. Der Aufbau der Versuche ist gegliedert in die Beschreibung des Experimentaufbaus, die Auswertung anhand von Metriken, der manuellen, stichprobenartigen Auswertung, der Klassifikation der Ergebnisse durch einen Menschen und der anschließenden Bewertung des durchgeführten Versuchs. Für alle Versuche werden folgende Hyperparameter eingesetzt:

**Allgemeine Konfigurationsparameter:**

| Name | Beschreibung | Wert |
|---|---|---|
| USE_CUDA | Boolean-Wert.<br>True = GPU verwenden<br>False = CPU verwenden | False |
| MAX_SENTENCE_LENGTH | Maximale Anzahl von Wörtern in einem Eingabesatz. | 21 |

**Netzwerk- Konfigurationsparameter:**

| Name | Beschreibung | Wert |
|---|---|---|
| learning_rate | Steuert, wie stark die Gewichte der Netzwerke in Bezug auf den Fehlergradienten angepasst werden. | 0.001 |
| teacher_forcing_ratio | Definiert, wie hoch der Korrekturanteil für die Netzausgaben einzelner Zeitschritte während der Backpropagation ist. | 0.6 |

| | | |
|---|---|---|
| clip | Definiert, wie stark der berücksichtigte Wirkungsgrad der einzelnen Fehlergradienten reduziert wird. | 5.0 |
| attn_model | Ausrichtungsmodell des Attention-Layers (Dot, General oder Concat) | General |
| hidden_size | Anzahl der Features für den Hidden-State | 300 |
| n_layers | Anzahl der verwendeten Netzwerkschichten. | 3 |
| dropout_p | Stärke der Dropout-Regulierungsmethode (Für das Deaktivieren von Neuronen, um eine Überanpassung zu verhindern) | 0.05 |

**Trainings- Konfigurationsparameter:**

| Name | Beschreibung | Wert |
|---|---|---|
| n_epochs | Anzahl der Epochen, welche für den Trainingsprozess verwendet werden. | 45000 |

Bei der Qualitätsbewertung von natürlicher Sprache handelt es sich, um ein nicht triviales Problem, welches nur schwer quantifizierbar ist. Die Qualität einer generierten Paraphrase ist daher nicht allein anhand der verwendeten Metriken zu ermitteln. Ein Beispiel dafür ist die folgende Ausgabe des Systems, welche ein durchaus akzeptables Ergebnis darstellt, jedoch einen sehr geringen BLEU-Score erhielt.

Loss: tensor(0.5428)
Cosine-similarity: 0.0
**BLEU_Score: 0.16508696268396486**
**Source** > heute wird ein regnerischer tag.
**Target** > höchstwahrscheinlich wird es heute daher niederschlag geben.
**Prediction** < es ist am heutigen tage mit niederschlag zu rechnen. <EOS>

Das gleiche Problem betrifft den Loss-Wert, welcher die Abweichung zwischen Vorhersage und Zielwert angibt. Im folgenden Beispiel wurde vom System ein hoher Fehlerwert für die Netzausgabe ermittelt, obwohl es sich um ein akzeptables Ergebnis handelt.

**Loss: tensor(5.8350)**
Cosine-similarity: 0.7514691493021793
BLEU_Score: 0.6926320811109014
**Source** >
ab dem mittag ist es fast vollständig bedeckt, aber die sonne ist durchgängig zu sehen.
**Target** >
ab dem mittag ist es fast vollständig bedeckt, aber wir sehen die sonne die ganze zeit.
**Prediction** <
gegen mittag ist es stark bewölkt, aber die sonne ist durchgängig zu sehen. <EOS>

Daher wird zusätzlich zu den Metriken und dem Loss-Wert, welche einen gewissen Qualitätsrahmen andeuten, eine manuelle Bewertung der Ergebnisqualität durchgeführt. Zu diesem Zweck werden stichprobenartig 100 Paraphrasen-paare aus den Ergebnissen ausgewählt und in folgende Klassen unterteilt:

- Fehlerfrei
- Sprachlich inkorrekt
- Fehlerhafter Kontext
- Zu niedrige Textvarianz

Ein Ergebnis wird nur genau einer Klasse zugeordnet. Sprachliche Fehler stehen über Kontextfehlern und Kontextfehler stehen über einer zu geringen Textvarianz. Im Folgenden wird für jede dieser Klassen ein Beispiel aufgeführt:

- Fehlerfrei:

**Source >** in der stadt <ort> ist es zu beginn des tages durchgehend bewölkt.
**Prediction <** in der stadt <ort> gibt es in den ersten stunden des tages kaum lücken in der wolkendecke. <EOS>

- Sprachlich inkorrekt:

**Source** > in der stadt <ort> ist der himmel zu tagesbeginn durch die fast komplett geschlossene wolkendecke kaum zu erkennen.
**Prediction** < in vormittag ist der himmel durchgehend bewölkt. <EOS>

| |
|---|
| ◆ Fehlerhafter Kontext:<br>**Source** > in der nacht ist der himmel fast sternenklar, und die temperaturen liegen bei <temperatur> bis maximal <temperatur><br>**Prediction** < in der frühe ist der himmel durchgehend bewölkt. <EOS> |
| ◆ Zu niedrige Textvarianz:<br>**Source** > während der nacht ist der himmel fast vollständig bewölkt, und die temperatur liegt bei <temperatur> bis maximal <temperatur><br>**Prediction** < während der nacht ist der himmel fast vollständig bewölkt und die temperatur bereich <temperatur> bis maximal <temperatur>. <EOS> |

## 6.1 Evaluation des Basisversuchs

In diesem Versuch soll untersucht werden, ob die Implementation des zuvor beschriebenen Systems die Anforderungen der Spezifikation erfüllen. Zudem soll der Basisversuch die Grundlage für weitere Experimente darstellen. Die Auswertung unterteilt sich in die Evaluation des Trainingsprozesses und der Evaluation trainierten Modelle anhand des Testdatensatzes. Zudem werden die Ausgaben der trainierten Modelle im Zuge der Qualitätssicherung einer manuellen, stichprobenartigen Untersuchung und Klassifikation unterzogen.

### 6.1.1 Versuchsaufbau

Zu Beginn des Experiments werden einige Wettertexte mit Hilfe der modifizierten Version des TextWriters von LangTec generiert. Diese Texte werden bereinigt und verwendet, um die benötigten Trainings- und Testdatensätze zu erzeugen. Der Trainingsdatensatz besteht aus 573 Paraphrasenpaaren. Darauf aufbauend wird ein Encoder- und ein Decoder-Modell trainiert. Die hierbei eingesetzten RNNs verwenden GRU-Zellen. Zum Schluss werden die trainierten Modelle auf Basis von 2255 unbekannten Testsätzen evaluiert. Aus der Evaluation resultieren die zu untersuchenden Systemausgaben und Metriken.

### 6.1.2 Bewertung des Trainingsprozesses

Abb. 11 zeigt die Evaluation des Trainingsprozesses. Die Abszisse dient der zeitlichen Einordnung und bildet die Trainingsepochen ab, während die Ordinate der Einordnung des Metrik-Scores dient. Der Graph zeigt den BLEU-Score als blaue, die Kosinus-Ähnlichkeit als rote und die Fehlerfunktion als grüne Kurve.

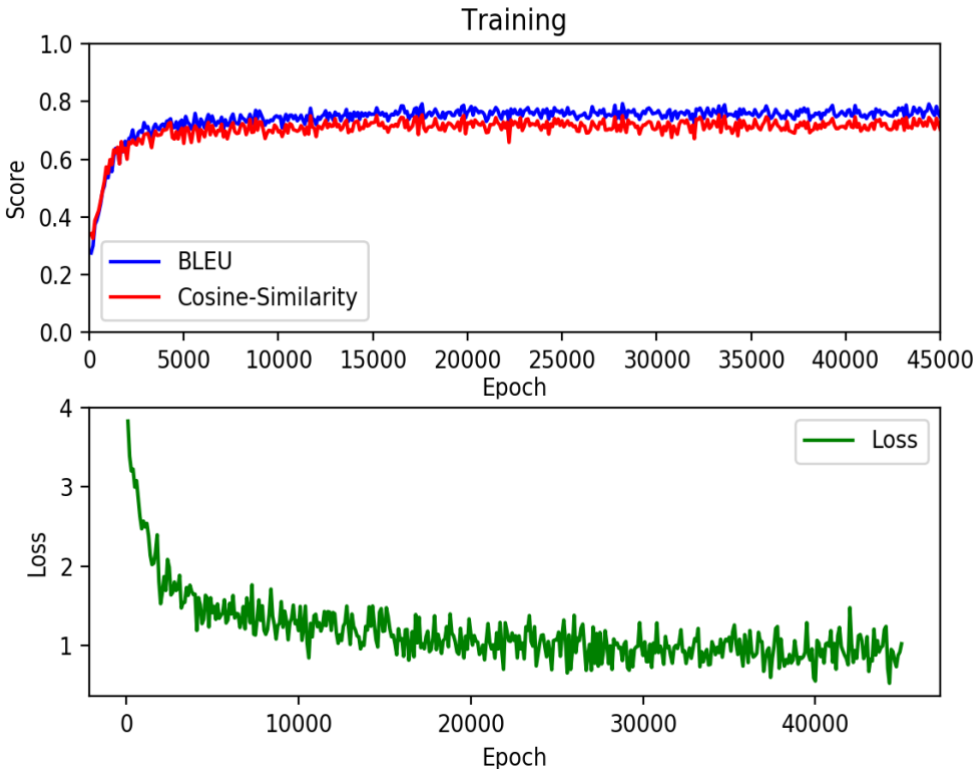



**Abbildung 11**: Evaluation des Trainings (Basisversuch)

Dieser Plot ist für sich genommen noch nicht sehr aussagekräftig, jedoch verdeutlicht er, dass sich der Score der Kosinus Ähnlichkeit und der BLEU-Score nach circa 4000 Epochen zwischen 0.6 und 0.8 einpendeln, während der Fehlerwert während des Trainings weiter reduziert wird. Die Kosinus-Ähnlichkeit zeigt, dass die Textvarianz zwischen Eingabesatz und Ausgabesatz im Mittel hinreichend groß ist. Der hohe BLEU-Score weist darauf hin, dass die generierten Paraphrasen die gewünschte Semantik abbilden. Der durchschnittliche NLLL-Wert wird während des Trainings beständig reduziert. Im Folgenden wird der Trainingsfortschritt anhand von Systemausgaben während des Trainings verdeutlicht:

**Epoche: 80**
Loss: tensor(1.9240)
Cosine-similarity: 0.0
BLEU_Score: 0.021247637560052053
**Source** > des nachts sinken die temperaturen weiter und erreichen temperaturen bis hin zu <temperatur>.
**Target** > heute nacht wird es weiterhin kälter und die tiefsttemperaturen erreichen werte von <temperatur>.
**Prediction** < ist ist ist ist der es es <EOS>

**Epoche: 1509**
Loss: tensor(1.9961)
Cosine-similarity: 0.40201512610368484
BLEU_Score: 0.4321227187565226
**Source** > es ist am heutigen tage mit niederschlag zu rechnen.
**Target** > am heutigen tag ist mit schauern zu rechnen.
**Prediction** < es wird heute mit wird es niederschlag <EOS>

**Epoche: 5430**
Loss: tensor(2.5321)
Cosine-similarity: 0.26726124191242434
BLEU_Score: 0.6220878649185788
**Source** > des nachts liegen die tiefsttemperaturen bei <temperatur>.
**Target** > nachts sinken die temperaturen auf <temperatur> – <temperatur>.
**Prediction** < zur nacht sinken die temperaturen auf frostige <temperatur>. <EOS>

**Epoche: 8391**
Loss: tensor(4.6841)
Cosine-similarity: 0.19245008972987526
BLEU_Score: 0.2845092008736424
**Source** > denken sie unbedingt an ihren regenschirm, es wird heute ein regnerischer tag.
**Target** > heute kann es voraussichtlich durchgehenden leichten regen geben.
**Prediction** < heute ist die wahrscheinlichkeit, dass es regnen sehr hoch. <EOS>

Diese Ausgaben des Trainingsprozesses verdeutlichen, den Trainingsfortschritt über Zeit. Sie zeigen, dass die gewählten Metriken in Kombination als grober Rahmen für die inhaltliche Qualität der generierten Paraphrasen dienen, jedoch keine Angaben über sprachliche Korrektheit treffen können.

### 6.1.3 Bewertung der trainierten Modelle

Der Testdatensatz wurde auf Basis der trainierten Modelle evaluiert und erzielte folgende Ergebnisse:

- *Durchschnittlicher BLEU-Score: 0.558417959120448*
- *Durchschnittliche Kosinus Ähnlichkeit: 0.5299852282478598*
- *Größe des verwendeten Vokabulars: 256*

Die Durchschnittswerte der Metriken erfüllen die Anforderungen aus der Spezifikation. Da die Kosinus Ähnlichkeit nur die Textvarianz eines spezifischen Paraphrasenpaares überprüft wurde zudem die Größe des verwendeten Vokabulars ermittelt, um die allgemeine Textvielfalt des Modells zu beschreiben. Da diese Daten die Qualität des Modells sehr allgemein Beschreiben und keinerlei Aussage über die Qualität einzelner Paraphrasen treffen folgt an dieser Stelle die manuelle Auswertung der Ergebnisse:

| | | | |
|---|---|---|---|
| ◆ Fehlerfrei: | 51 | ◆ Sprachlich inkorrekt: | 12 |
| ◆ Fehlerhafter Kontext: | 28 | ◆ Zu niedrige Textvarianz | 9 |

51 von 100 Paraphrasen sind sowohl sprachlich als auch inhaltlich korrekt und verfügen über eine hinreichend große Textvarianz in Bezug auf den Ausgangstext. Die Ergebnisse zeigen, dass das zu beschreibende Wetterphänomen, zwar mit hoher Genauigkeit erkannt wird, es dem Modell jedoch schwer fällt bestimmte natürlich sprachliche Kontextelemente zu erkennen. Beispielsweise werden Windrichtungen verwechselt oder es wird eine starke Bewölkung, statt einer leichten Bewölkung beschrieben.

**Beispiel aus der Auswertung:**
**Source** > am abend sinkt das thermometer auf werte zwischen <temperatur> und <temperatur> es weht ein leichter wind aus nordost.
**Prediction** < gegen abend fällt das thermometer auf werte zwischen <temperatur> und <temperatur> es weht ein geringer wind aus südwesten. <EOS>

Eine weitere Erkenntnis ist, dass die Verteilung der Trainingsdaten eine wichtige Rolle spielt, daher sind die Ergebnisse für bestimmte Wetterphänomene, welche häufig in den Daten vorkommen deutlich besser als für jene die seltener in den Daten vorkommen.

## 6.2 Experiment 001: Augmentierung der Daten

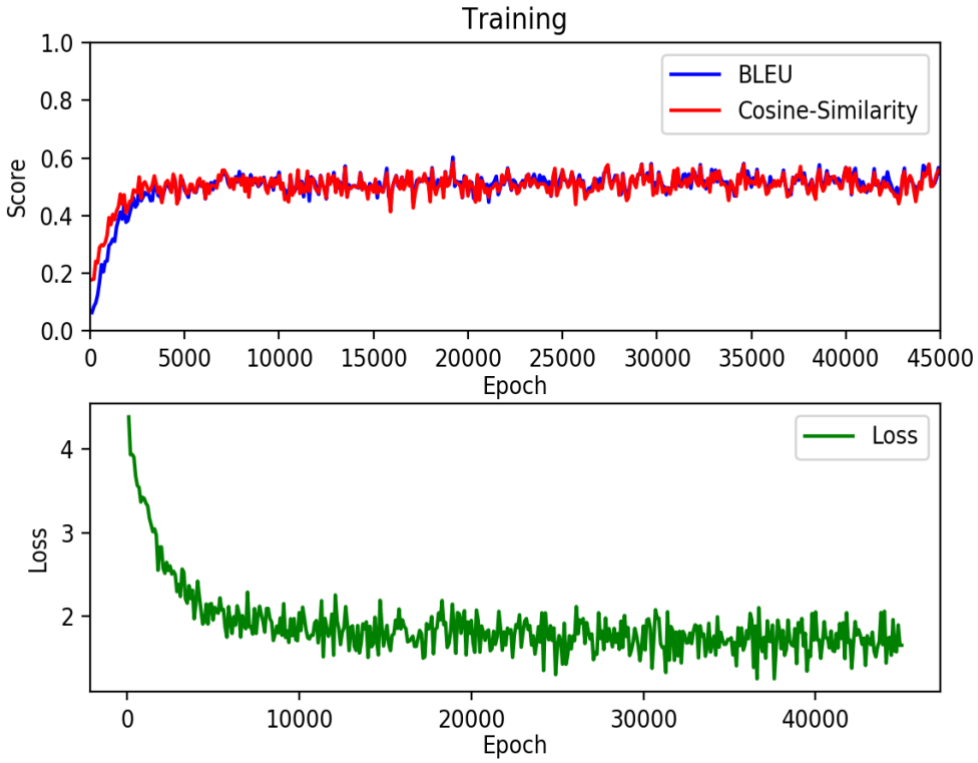
Aufbauend auf dem Basisversuch wurde die Datenbasis in diesem Experiment manuell durch alternative Texte erweitert. Hierbei wurde besonders für Wetterphänomene, welche im Basisversuch schlechte Ergebnisse erzielten weitere Alternativen hinzugefügt. Das Experiment soll die Textvielfalt des trainierten Modells erhöhen und die Verteilung einzelner Wetterphänomene in den Trainingsdaten regulieren, damit alle Phänomene ausgeglichen trainiert werden.

### 6.2.1 **Versuchsaufbau**

Zu Beginn des Experiments werden einige Wettertexte mit Hilfe der modifizierten Version des TextWriters von LangTec generiert. Zusätzlich werden manuell erstellte, alternative Wettertexte hinzugefügt, welche sich inhaltlich und sprachlich an den generierten Texten orientieren. Diese Texte werden anschließend bereinigt und verwendet, um die benötigten Trainings- und Testdatensätze zu erzeugen. Der augmentierte Trainingsdatensatz besteht aus 3248 Paraphrasenpaaren. Darauf aufbauend wird ein Encoder- und ein Decoder-Modell trainiert. Die hierbei eingesetzten RNNs verwenden GRU-Zellen. Zum Schluss werden die trainierten Modelle auf Basis von 2293 unbekannten Testsätzen evaluiert. Aus der Evaluation resultieren die zu untersuchenden Systemausgaben und Metriken.

### 6.2.2 Bewertung des Trainingsprozesses

Abb. 12 zeigt die Evaluation des Trainingsprozesses analog zum Basisversuch.




Das Kurven verhalten sich analog zum Basisversuch, jedoch ist der Durchschnittswert für den BLEU-Score und die Kosinus Ähnlichkeit gesunken und bewegt sich nun zwischen 0.4 und 0.5, statt wie zuvor zwischen 0.6 und 0.8. Die Textvarianz zwischen Ausgangssatz und Vorhersage ist durch die hinzugefügten Texte deutlich gestiegen. Wie zu erwarten sinkt jedoch auch der durchschnittliche BLEU-Score, da die Kopplung zwischen den beiden Metriken sehr hoch ist. Dies muss jedoch nicht zwangsweise bedeuten, dass die inhaltliche Qualität der Paraphrasen abgenommen hat. Die folgende Ausgabe aus der Evaluation zeigt, dass der BLEU-Score bei einer hohen Textvarianz hinsichtlich der Kontextidentifikation an seine Grenzen stößt.

**Abbildung 12**: Evaluation des Trainings (Experiment 001)

Cosine-similarity: 0.13453455879926252
BLEU_Score: 0.22020404924648757
**Source** > zu beginn des tages ist im <ort> mit starker bewölkung zu rechnen.
**Prediction** < in der stadt <ort> ist die wolkendecke in der frühe fast vollständig geschlossen. <EOS>

### 6.2.3 Bewertung der trainierten Modelle

Der Testdatensatz wurde auf Basis der trainierten Modelle evaluiert und erzielte folgende Ergebnisse:

- *Durchschnittlicher BLEU-Score: 0.5187144831148106*
- *Durchschnittliche Kosinus-Ähnlichkeit: 0.5108791495129907*
- *Größe des verwendeten Vokabulars: 407*

Die Durchschnittswerte der Metriken erfüllen die Anforderungen aus der Spezifikation. Im Vergleich zum Basisversuch haben sich die Werte für die Evaluation der Testdaten nicht signifikant verändert. Die Größe des verwendeten Vokabulars ist jedoch deutlich größer und spricht für eine größere Textvielfalt des Modells im Allgemeinen. Analog zum Basisversuch folgen die Ergebnisse der manuellen Auswertung:

| | | | |
|---|---|---|---|
| Fehlerfrei: | 54 | Sprachlich inkorrekt: | 9 |
| Fehlerhafter Kontext: | 32 | Zu niedrige Textvarianz | 5 |

54 von 100 Paraphrasen sind sowohl sprachlich als auch inhaltlich korrekt und verfügen über eine hinreichend große Textvarianz in Bezug auf den Ausgangstext. Trotz der Augmentierung der Daten hat sich das Ergebnis der Evaluation der Testdaten nicht signifikant verändert. Es besteht immer noch das Problem, dass Phänomene zwar oft erkannt werden, jedoch die Details zu den Phänomenen häufig falsch gedeutet werden. Zudem ist zu beobachten, dass die Vorhersagegenauigkeit für einzelne Wetterphänomene bei steigender Komplexität weiter sinkt. Da das System auf für größere Domänen eingesetzt werden soll, muss ein alternativer Trainingsworkflow verwendet werden, um die Ergebnisqualität des Systems von der wachsenden Komplexität von Subdomänen zu entkoppeln. Als mögliche Maßnahme könnten die Subdomänen einzeln in separaten Modellen trainiert werden. Es ist anzunehmen, dass sich dadurch die Paraphrasenqualität für die einzelnen Subdomänen deutlich verbessert. Die Modelle könnten alle auf einen gemeinsamen Wortschatz zugreifen wie es bisher bereits die Modelle des Encoders und des Decoders tun. Während der Generierung müsste jedoch mit Hilfe eines Klassifizierers entschieden werden, zu welcher Domäne der Eingabetext gehört, um das entsprechende Modell für die Generierung zu verwenden.

## 6.3 Experiment 002: LSTM-Zellen statt GRU-Zellen

Aufbauend auf Experiment 001 wird in diesem Experiment untersucht, welche Auswirkung der Austausch der GRU-Zellen in den RNNs des Encoders und Decoders durch LSTM-Zellen hat. LSTM-Zellen verwenden im Gegensatz zu GRU-Zellen einen speziellen Zellen-Status und verwenden diesen statt des Hidden-States um Informationen zu transferieren. GRU-Zellen verwenden weniger Tensor-Operationen. Daher lassen sich GRU-Zellen in der Regel schneller trainieren, als LSTM-Zellen. GRU-Zellen erreichen häufig bessere Ergebnisse auf kleineren Trainingsdatensätzen, jedoch eignen sich LSTM-Zellen besser für das Verarbeiten von langen Sequenzen und dem Modellieren von Long-Distance-Relations. [Ngu18]

Da der Trainingsdatensatz zum Großteil aus langen Sequenzen bis zu 21 Wörtern je Satz besteht, soll in diesem Experiment erfasst werden, ob sich das Ergebnis durch das Training mit LSTM-Zellen signifikant verändert.

### 6.3.1 Versuchsaufbau

Das Experiment baut auf demselben Trainings- und Testdatensatz auf wie Experiment 001. Der Trainingsdatensatz besteht daher aus 3248 Paraphrasenpaaren. Darauf aufbauend wird ein Encoder- und ein Decoder-Modell trainiert. Anders als in den vorherigen Experimenten werden für die verwendeten RNNs LSTM-Zellen statt GRU-Zellen eingesetzt. Zum Schluss werden die trainierten Modelle auf Basis von 2293 unbekannten Testsätzen evaluiert. Aus der Evaluation resultieren die zu untersuchenden Systemausgaben und Metriken.

### 6.3.2 Bewertung des Trainingsprozesses

Abb. 13 zeigt die Evaluation des Trainingsprozesses analog zu vorherigen Versuchen. Im Vergleich mit Experiment 001 lassen sich keine signifikanten Veränderungen im Kurvenverhalten feststellen. Während der Auswertung des Trainingsprozesses konnten keine nennenswerten Veränderungen festgestellt werden.

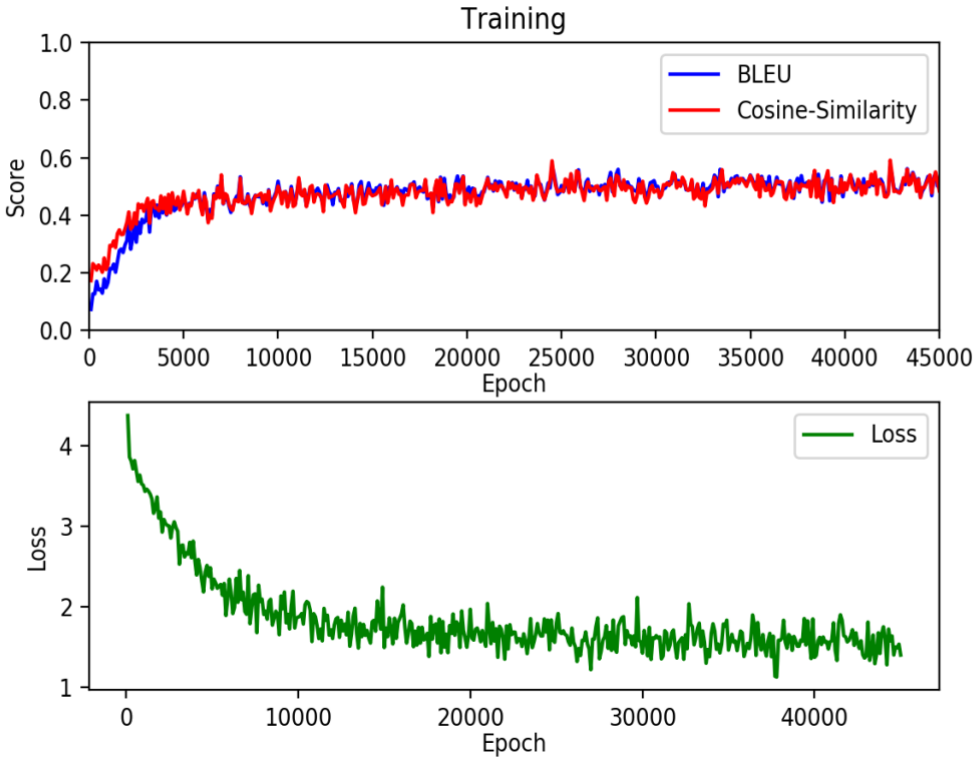



**Abbildung 13**: Evaluation des Trainings (Experiment 002)

### 6.3.3 Bewertung der trainierten Modelle

Der Testdatensatz wurde auf Basis der trainierten Modelle evaluiert und erzielte folgende Ergebnisse:

- *Durchschnittlicher BLEU-Score: 0.5166554757710623*
- *Durchschnittliche Kosinus-Ähnlichkeit: 0.4925200963245295*
- *Größe des verwendeten Vokabulars: 407*

| | | | |
|---|---|---|---|
| Fehlerfrei: | 52 | Sprachlich inkorrekt: | 8 |
| Fehlerhafter Kontext: | 34 | Zu niedrige Textvarianz | 6 |

Es konnten keinerlei nennenswerten Abweichungen von den Ergebnissen aus Experiment 001 festgestellt werden.

## 6.4 Experiment 003: Forcieren der Varianz

In diesem Experiment soll mit Hilfe von zwei methodisch unterschiedlichen Ansätzen versucht werden, auf Basis der trainierten Modelle mehrere Paraphrasen als Ausgaben für einen Eingabesatz zu erzeugen.

### 6.4.1 Versuchsaufbau

Das Experiment baut auf demselben Trainings- und Testdatensatz auf wie Experiment 001. Der Trainingsdatensatz besteht daher aus 3248 Paraphrasenpaaren. Darauf aufbauend wird ein Encoder- und ein Decoder-Modell trainiert. Die hierbei eingesetzten RNNs verwenden GRU-Zellen. Zum Schluss werden die trainierten Modelle auf Basis von 2293 unbekannten Testsätzen evaluiert. Im ersten Ansatz des Experiments wird ein Skript eingesetzt, welches die zu verarbeiteten Tensor-Variablen in einer Schleife evaluiert. Hierbei wird in allen Folgeiterationen die zuvor generierte Paraphrase als Eingabe verwendet, um eine neue Paraphrase zu generieren. Im Ansatz 2 wird die Embedding-Matrix der Eingabesequenz während der Evaluation durch das Addieren von pseudozufälligen Rauschwerten manipuliert. Die Ergebnisse der Evaluation sind Systemausgaben, welche hinsichtlich ihrer Textvarianz untersucht werden.

### 6.4.2 Ansatz 1: Rekursive Ergebnisparaphrasierung

Die Ergebnisse der Evaluation auf Basis der trainierten Modelle sind deterministisch und erzeugen für einen spezifischen Eingabetext eine spezifische Paraphrase. In diesem Ansatz wird versucht alternative Paraphrasen für denselben Eingabetext zu erzeugen, indem die Ergebnisparaphrasen rekursiv als Eingabe für eine weitere Evaluation verwendet werden. Die folgende Systemausgabe repräsentiert das Ergebnis dieses Versuches:

| |
|---|
| Cosine-similarity: 0.2519763153394848<br>BLEU_Score: 0.2532329560537093<br>**Source** > es ist am heutigen tage mit niederschlag zu rechnen.<br>**Prediction** < es wird daher mit hoher wahrscheinlichkeit regnen. <EOS> |
| Cosine-similarity: 0.6546536707079772<br>BLEU_Score: 0.42608910005509176<br>**Source** > es wird daher mit hoher wahrscheinlichkeit regnen.<br>**Prediction** < es wird daher mit hoher wahrscheinlichkeit niederschlag geben, möglicherweise durchgehenden leichten regen. <EOS> |

| |
|---|
| Warning: Cosine-Similarity is 1.0000000000000002 but it should be lower than 0.7<br>Cosine-similarity: 1.0000000000000002<br>BLEU_Score: 0.9906092887336139<br>**Source** > es wird daher mit hoher wahrscheinlichkeit niederschlag geben, möglicherweise durchgehenden leichten regen.<br>**Prediction** < es wird daher mit hoher wahrscheinlichkeit niederschlag geben, möglicherweise durchgehenden leichten regen. <EOS> |
| Warning: Cosine-Similarity is 1.0000000000000002 but it should be lower than 0.7<br>Cosine-similarity: 1.0000000000000002<br>BLEU_Score: 0.9906092887336139<br>**Source** > es wird daher mit hoher wahrscheinlichkeit niederschlag geben, möglicherweise durchgehenden leichten regen.<br>**Prediction** < es wird daher mit hoher wahrscheinlichkeit niederschlag geben, möglicherweise durchgehenden leichten regen. <EOS> |

Aus den Ergebnissen der Auswertung geht hervor, dass sich die Metrik-Scores mit jeder Iteration erhöhen und das System schnell in einen Zustand gerät, an dem es den Eingabetext unverändert als Ergebnis zurückgibt. Dieser Lösungsansatz erweist als zu naiv, um die Varianz der Systemausgaben signifikant zu forcieren.

### 6.4.3 Ansatz 2: Manipulation der Eingabesequenzen im Encoder durch Pseudozufallsrauschen

Bei diesem Ansatz wird während der Evaluation ein pseudozufälliger Rauschwert im Bereich von -0.01 bis 0.01 auf einen Teil der Tensoren der Embedding-Matrix der Eingabesequenz addiert. Dies dient dem Zweck, dass derselbe Eingabesatz durch die geringen Abweichungen der Werte auf unterschiedliche Weise vom Decoder-Netz interpretiert wird. Die folgende Systemausgabe repräsentiert das Ergebnis dieses Versuches:

| |
|---|
| Cosine-similarity: 0.42640143271122094<br>BLEU_Score: 0.42178106147966354<br>**Source** > mittags ist es stark bewölkt, aber die sonne zeigt sich durchgängig.<br>**Prediction** < ab dem mittag ist es fast vollständig bedeckt, aber wir sehen die sonne die ganze zeit. <EOS> |
| Cosine-similarity: 0.5393598899705937<br>BLEU_Score: 0.48300754225110437<br>**Source** > mittags ist es stark bewölkt, aber die sonne zeigt sich durchgängig.<br>**Prediction** < um die mittagszeit ist der himmel fast vollständig bewölkt, aber die sonne ist durchgängig zu sehen. <EOS> |

| |
|---|
| Cosine-similarity: 0.42640143271122094<br>BLEU_Score: 0.42178106147966354<br>**Source** > mittags ist es stark bewölkt, aber die sonne zeigt sich durchgängig.<br>**Prediction** < ab dem mittag ist es fast vollständig bedeckt, aber wir sehen die sonne die ganze zeit. <EOS> |
| Cosine-similarity: 0.23354968324845687<br>BLEU_Score: 0.33695669496644476<br>**Source** > mittags ist es stark bewölkt, aber die sonne zeigt sich durchgängig.<br>**Prediction** < gegen mittag ist die sonne bei vollständig bewölktem himmel in <ort> dennoch ständig zu sehen. <EOS> |

Aus den Ergebnissen der Evaluation geht hervor, dass das System mit Hilfe des angewendeten Lösungsansatzes in der Lage ist die Varianz der Ausgabe in einem begrenzten Maße zu forcieren. Durch die Manipulation der Eingabesequenzen kann es jedoch vorkommen, dass die ursprüngliche Bedeutung der Eingabesequenz verfälscht wird und sich dies auf die vom Decoder vorhergesagte Paraphrase überträgt. Zudem ist der Raum für Varianz durch verschiedene Faktoren wie beispielsweise die Größe des verwendeten Vokabulars beschränkt. Das System kann daher nur endlich viele korrekte, alternative Ergebnisse generieren.

# 7 Fazit

In diesem Kapitel werden die Inhalte dieser Bachelorarbeit, sowie die daraus gewonnen Erkenntnisse abschließend zusammengefasst und bewertet. Darüber hinaus wird ein Ausblick auf mögliche zukünftige Erweiterungen und Anwendungsfälle des entwickelten Systems gegeben.

## 7.1 Zusammenfassung

Im Rahmen dieser Arbeit wurde ein System zum Paraphrasieren domänenspezifischer Texte mit neuronalen Netzen entwickelt. Dieses Projektvorhaben entstand in der Zusammenarbeit mit dem Hamburger Unternehmen LangTec, welches als Technologie- und Dienstleistungsanbieter im Bereich der semantischen Textanalyse, computerlinguistischen Beratung und Lokalisierung tätig ist. Das Ziel war es, ein System zu entwickeln, welches als Ergänzung zu bestehenden NLG-Prozessen eingesetzt werden kann, um unter Verwendung von Methoden und Konzepten aus dem Bereich der künstlichen Intelligenz alternative Texte zu generieren. Aus der Analyse gingen die spezifizierten Anforderungen hervor, welche an dieses System gestellt werden. Es wurde beschlossen, den konzeptionellen Beweis für die Funktionalität des Systems anhand der Domäne von Wettertexten zu erbringen. Hierfür stellte das Unternehmen LangTec Trainingsdaten aus ihrem Textgenerierungs-Tool „Textwriter“ bereit, welche im späteren Verlauf manuell erweitert wurden. Als Kernkomponente des Systems wurde das Seq2seq Framework, basierend auf einer Encoder-Decoder-Architektur mit zwei RNNs eingesetzt. Das System wurde auf Basis der getroffenen Designentscheidungen implementiert und anschließend durch verschiedene Experimente inkrementell erweitert. Als Maßnahme zur Qualitätssicherung wurden verschiedene Metriken eingesetzt, um die Ergebnisse der Evaluation in Relation zu setzen und einen Maßstab für die Ergebnisqualität zu erhalten. Hierfür wurde der BLEU-Score, die Kosinus Ähnlichkeit und Verlustfunktion (negative log likelihood loss) eingesetzt. Da es sich bei der Generierung von Paraphrasen jedoch um ein schwer quantifizierbares Problem handelt, wurde darüber hinaus eine manuelle Auswertung durchgeführt.

Es erwies sich als schwierig, ein Modell für eine gesamte Domäne zu trainieren, da nicht nur genügend Daten für alle Subdomänen bereitgestellt werden müssen, sondern auch die anteilige Verteilung der Daten auf die einzelnen Subdomänen eine tragende Rolle spielt. Die

trainierten Modelle sind in der Lage für etwa die Hälfte der gegebenen Testdaten hochwertige Paraphrasen zu erzeugen. Ein häufiges Problem war es, dass bestimmte Subdomänen aufgrund unvorteilhafter Verteilung in den Trainingsdaten deutlich schlechtere Ergebnisse lieferten als andere Subdomänen. Zudem zeigte sich, dass das System oftmals Schwierigkeiten mit der Zuordnung von Word-Embeddings hat, welche eine geringe Distanz untereinander haben. Dies führte dazu, dass Subdomänen zwar erkannt wurden, die dazugehörigen Details jedoch teilweise verwechselt wurden. Beispielweise wurden in den generierten Paraphrasen zum Teil Details zu Windrichtungen oder Bewölkungsgrade falsch wiedergegeben. Eine weitere wichtige Erkenntnis ist, dass es schwierig ist, ein geeignetes Maß zum Quantifizieren natürlich sprachlicher Modelle zu finden. Das verwendete Seq2Seq Framework erwies sich als außerordentlich flexibel. Das System kann aufgrund der niedrigen Kopplung der Encoder-Decoder-Architektur für unterschiedliche Anwendungsfälle eingesetzt werden. Allein durch den Austausch der Trainings- und Testdaten könnte das System beispielsweise zum Übersetzen natürlicher Sprache, oder als Chatbot verwendet werden. Aufbauend auf den gewonnenen Erkenntnissen wird im Folgenden ein Ausblick auf mögliche Erweiterungen des Systems gegeben.

## 7.2 Ausblick

In diesem Kapitel wird ein Ausblick auf eine mögliche Weiterentwicklung des Systems gegeben. Um dem Problem der Datenverteilung für die Subdomänen, dessen Komplexität bei steigender Anzahl von Subdomänen weiter steigt entgegenzuwirken, könnte ein Experiment unter der Verwendung des Divide-And-Conquer-Verfahrens durchgeführt werden. Hierbei könnte für jede Subdomäne ein eigenes Modell trainiert werden. Anschließend könnte der Ausgangstextes während des Generierung-Prozesses mit Hilfe eines Klassifizierers einer Subdomäne zugeordnet werden und auf Basis des entsprechenden Modells Paraphrasen generieren. Der Klassifizierer wäre zwar in diesem Fall der Flaschenhals des Systems, jedoch ist das Klassifikationsproblem, welches hierbei gelöst werden soll deutlich weniger komplex, als das kontextbasierte Generieren natürlicher Sprache. Das System könnte im späteren Verlauf auf weiteren Domänen mit unterschiedlichen Komplexitätsgraden trainiert werden. Um das System automatisiert an einen NLG Prozess anzubinden ist es notwendig, eine geeignete Bewertungsmetrik für die generierten Paraphrasen zu finden, da die hier verwendeten Metriken nicht ausreichen, um die Qualität einer Paraphrase ohne manuelle Auswertung hinreichend zu bewerten. Ein weiteres interessantes Experiment wäre die Verwendung von BERT. Hierbei handelt es sich um ein bereits trainiertes Word-Embedding-Modell, welches im Jahr 2018 große Erfolge im Bereich NLP erzielte. Die Generierung von alternativem Text ist ein wichtiger NLG-Prozess, welcher auch in der Zukunft in vielen Bereichen Anwendung finden könnte. Ein System zur Generierung von Paraphrasen könnte ein erster Schritt in Richtung vollautomatisierter Content-Generierung auf Basis einer maschinellen Kontextinterpretation sein.

# Anhang 1:

## Technische Dokumentation

In der technischen Dokumentation werden die implementierten Klassen, Methoden und Funktionen des Systems beschrieben.

### Funktionen der Datenaufbereitung

```
def cleanup_text(text):
    """
    Dient dem Bereinigen der Texte, welche von den
    Jinja2 Templates des LangTec Textwriter generiert wurden.
    Alle Nicht-ASCII Zeichen werden durch UTF-8 Zeichen ersetzt.
    Dynamische Daten wie Orte, Temperaturen und Monate werden
    durch Tokens ersetzt.
    :param text: Der unverarbeitete Templatetext als String
    :return: Bereinigter Text als String
    """
```

```
def extract_wetter_texts(file_path):
    """
    Dient dem Einlesen von
    Templatetexten aus Text-Dateien.
    :param file_path: Der Pfad zur Textdatei
    :return: Der extrahierte und geparste Text als String.
    """
```

```
def write_to_file(train_textfile_path, test_textfile_path):
    """
    Schreibt die Trainingsdaten Text-Paare und
    die Testsätze in Textdateien.
    :param train_textfile_path: Pfad zur Textdatei
    der Trainingsdaten
    :param test_textfile_path: Pfad zur Textdatei
    der Testdaten
    :return:
    """
```

```
def extract_template_paraphrases():
    """
    Extrahiert Paraphrasen aus Templatetexten,
    welche für dieselben Datensätze generiert wurden.
    Die Funktion ermittelt die regionalen Texte für
    einen Datensatz und übergibt diese an
    collect_template_pairs(region_path, textfiles)
    :return:
    """
```

```
def collect_template_pairs(region_path, textfiles):
    """
    20% der Texte werden als Testdaten in der Test-Textdatei
    gespeichert. Die restlichen 80% der regionalen Texte für
    einen Datensatz werden kombiniert und zu Paraphrasenpaaren
    zusammengefügt. Die Paraphrasenpaare werden nach einer
    Überprüfung in die Textdatei für Trainingsdaten geschrieben.
    :param region_path: Pfad zum Regionsverzeichnis der Texte für
    eine Region
    :param textfiles: Alle für diese Region generierten Texte für
    einen Datensatz.
    :return:
    """
```

```
def extract_experimental_paraphrases():
    """
    20% der manuell erstellten Texte werden als Testdaten
    in der Test-Textdatei gespeichert. Die restlichen 80%
    der experimentellen Texte werden für jedes zu beschreibende
    Wetterphänomen kombiniert und zu Paraphrasenpaaren
    zusammengefügt. Die Paraphrasenpaare werden nach einer
    Überprüfung in die Textdatei für Trainingsdaten geschrieben.
    :return:
    """
```

## Language Klasse

```
class Lang:
    """
    Klasse zum Erstellen einer Sprache auf Basis der Texte in den
    Trainingsdaten. Indizes werden Wörtern zugewiesen und umgekehrt.
    Alle Wörter werden in einem Vokabular gesammelt und das
    Vorkommen der einzelnen Wörter wird gezählt.
    Initialisierungs-Parameter:
   :param name: Ein String der den Namen der Sprache
    repräsentiert
    """
```

```
def Lang.index_words(self, sentence):
    """
    Weist jedem Wort im übergebenen Satz einen Index zu.
    :param sentence: Ein String mit Wörtern, welche durch
    Leerzeichen getrennt sind.
    :return:
    """
```

```
def Lang.index_word(self, word):
    """
    Indiziert ein einzelnes Wort.
    :param word: Ein String der ein einzelnes Wort enthält.
    :return:
    """
```

```
def create_language(lang_name):
    """
    Erstellt ein Language-Objekt auf Basis der Trainingsdaten und
    sammelt die Paraphrasenpaare in einer Liste.
    :param lang_name: Ein String der den Namen der Sprache
    repräsentiert
    :return: Gibt ein Language-Objekt und eine 2-dimensionale
    Liste mit Paraphrasenpaaren zurück.
    """
```

```
def indices_from_sentence(lang, sentence):
    """
    Sammelt Indizes des Language-Objekts für jedes Wort im
    übergebenen Satz.
    :param lang: Language-Objekt
    :param sentence: Ein String mit Wörtern, welche durch
    Leerzeichen getrennt sind.
    :return: Gibt eine Liste von Indizes zurück, einen für
    jedes Wort im übergebenen Satz
    """
```

```
def variable_from_sentence(lang, sentence):
    """
    Erstellt eine PyTorch-Tensorvariable auf Basis
    Des übergebenen Satzes.
    :param lang: Language-Objekt
    :param sentence: Ein String mit Wörtern, welche durch
    Leerzeichen getrennt sind.
    :return: Gibt eine PyTorch-Tensorvariable zurück
    """
```

```
def variables_from_pair(lang, pair):
    """
    Erstellt zwei PyTorch-Tensorvariablen für den Ausgangs- und
    Zielsatz eines Paraphrasenpaars.
    :param lang: Language-Objekt
    :param pair: Eine Liste mit zwei String-Elementen
    :return: Gibt zwei PyTorch-Tensorvariable zurück
    """
```

## Funktionen zum Berechnen und Darstellen von Metriken

```
def get_bleu_score(s1, s2):
    """
    Berechnet den BLEU-Wert zwischen dem Eingabetextes und dem
    vorhergesagten Ausgabetextes
    :param s1: Eingabetext
    :param s2: Vorhergesagter Ausgabetext
    :return: Gibt den BLEU-Wert zurück
    """
```

```
def get_cosine(s1, s2):
    """
    Berechnet die Kosinus-Ähnlichkeit zwischen dem Eingabetextes und
    dem vorhergesagten Ausgabetextes
    :param s1: Eingabetext
    :param s2: Vorhergesagter Ausgabetext
    :return: Gibt den Wert der Kosinus-Ähnlichkeit zurück
    """
```

```
def text_to_vector(text):
    """
    Bildet einen Text auf einen Vektor ab, indem das Vorkommen der
    einzelnen Wörter gezählt wird.
    :param text: Text als String
    :return: Gibt einen Vektor als Counter-Objekt zurück
    """
```

```
def plot_scores(n_epochs, epochs_to_plt, bleu_scores, cosine_scores, loss=None):
    """
    Visualisiert die Metriken mit Hilfe von PyPlot
    :param n_epochs: Anzahl der Epochen während des Trainings
    :param epochs_to_plt: Liste der Epochen, an denen
    ein gemittelter Datenpunkt eingetragen wird
    :param bleu_scores: Liste durchschnittlichen Bleu-Werte
    :param cosine_scores: Liste durchschnittlichen
    Kosinus-Ähnlichkeits-Werte
    :param loss: Liste der durchschnittlichen Fehlerwerte
    :return:
    """
```

## Encoder

```
class EncoderRNN(nn.Module):
    """
    Klasse zum Erstellen eines RNN-Netzes für den Encoder
    als Subklasse der PyTorch nn.Module-Klasse.
    Initialisierungs-Parameter:
    :param input_size: Anzahl der erwarteten Feature für
    die Netzeingabe
    :param hidden_size: Anzahl der Feature für den Hidden-State
    :param n_layers: Anzahl der Netzschichten. Default = 1
    """
```

```
def EncoderRNN.forward(self, word_inputs, hidden):
    """
    Forwarding Methode zum Propagieren durch das RNN
    in jedem Zeitschritt.
    :param word_inputs: Tensor der Eingabesequenz.
    :param hidden: Hidden-State aus Initialisierung oder der
    der vorherigen Netzausgabe
    :return: Gibt einen Kontext-Vektor und einen Hidden-State zurück
    """
```

```
def EncoderRNN.init_hidden(self):
    """
    Initialisiert auf Basis der Anzahl der Netzwerkschichten
    und der Anzahl der Hidden-Features einen Hidden-State als
    Pytorch Variable und gibt diesen zurück.
    :return: Hidden-State
    """
```

## Decoder

```
class Attn(nn.Module):
    """
    Klasse zum Erstellen eines RNN-Netzes als Subklasse der
    PyTorch nn.Module-Klasse, um auf Basis des
    Encoder-Outputs die Attention-Gewichte zu berechnen.
    Initialisierungs-Parameter:
    :param method: Ausrichtungsmodell (dot, general oder concat)
    :param hidden_size: Anzahl der Features für den Hidden-State
    """
```

```
def Attn.forward(self, hidden, encoder_outputs):
    """
    Forwarding-Methode zum Propagieren durch das RNN
    in jedem Zeitschritt.
    :param hidden: Hidden-State aus der Initialisierung oder
    der vorherigen Netzausgabe
    :param encoder_outputs: Kontextvektor und Hidden-State
    des Encoders
    :return:
    """
```

```
def Attn.score(self, hidden, encoder_output):
    """
    Berechnung der Energien (unnormalisierte Wahrscheinlichkeiten)
    des entsprechenden Ausrichtungsmodells für jeden Hidden-State
    des Decoders.
    :param hidden: Hidden-State des Decoders
    :param encoder_output: Netzausgabe des Encoders
    :return: Energie des Ausrichtungsmodells
    """
```

```
class AttnDecoderRNN(nn.Module):
    """
    Klasse zum Erstellen eines RNN-Netzes für den Decoder
    als Subklasse der PyTorch nn.Module-Klasse.
    Initialisierungs-Parameter:
    :param attn_model: Ausrichtungsmodell (dot, general oder concat)
    :param hidden_size: Anzahl der Features für den Hidden-State
    :param output_size: Anzahl der Features der Netzausgabe
    :param n_layers: Anzahl der Netzwerkschichten. Default = 1
    :param dropout_p: Stärke der Dropout-Regulierungsmethode
    """
```

```
def AttnDecoderRNN.forward(self, word_input, last_context, last_hidden, encoder_outputs):
    """
    Forwarding-Methode zum Propagieren durch das RNN
    in jedem Zeitschritt.
    :param word_input: Tensor der Eingabesequenz.
    :param last_context: Kontextvektor aus der letzten
    Netzausgabe des Decoders.
    :param last_hidden: Hidden-State aus Initialisierung oder der
    der vorherigen Netzausgabe
    :param encoder_outputs: Netzausgabe des Encoders
    :return: Netzausgabe des Decoders, finaler Kontextvektor,
    finaler Hidden-State, finale Attention-Gewichte
    """
```

## Training

```
def train(input_variable, target_variable, encoder, decoder, encoder_optimizer, decoder_optimizer,
criterion):
    """
    Trainiert die Gewichte des Encoder- und Decodernetzes
    auf Basis eines übergebenen Paraphrasenpaares.
    :param input_variable: PyTorch Tensor-Variable
    für einen Eingabesatz
    :param target_variable: PyTorch Tensor-Variable
    für einen Zielsatz
    :param encoder: EncoderRNN-Objekt
    :param decoder: AttnDecoderRNN-Objekt
    :param encoder_optimizer: PyTorch Optimizer-Objekt
    :param decoder_optimizer: PyTorch Optimizer-Objekt
    :param criterion: PyTorch Kriterium-Objekt für die
    Verlustfunktion (negative log likelihood loss)
    :return: Gibt den Verlustwert der Fehlerfunktion und die
    Netzmodelle des Encoders und Decoders zurück
    """
```

```
def as_minutes(s):
    """
    Rechnet Sekunden in Minuten um
    :param s: Sekunden als Int
    :return: Minuten und Sekunden als String
    """
```

```
def time_since(since, percent):
    """
    Berechnet die für das Training benötigte Zeit
    :param since: Startzeit
    :param percent: Prozentangabe als Int
    :return: Verstrichene Zeit als String
    """
```

## Evaluation

```
def evaluate(sentence, target_sentence=None, train=False, encoder=None, decoder=None):
    """
    Evaluiert einen übergebenen Satz auf Basis der trainierten
    Modelle und der Sprache.
    :param sentence: Ein String mit Wörtern, welche durch
    Leerzeichen getrennt sind.
    :param target_sentence: Optionaler Zielsatz zum Evaluieren
    von Paraphrasen im Trainingsprozesses.
    :param train: Boolean, der definiert, ob es sich um die
    Evaluation
    des Trainings handelt.
    :param decoder: Decoder-Objekt für das Training.
    :param encoder: Encoder-Objekt für das Training.
    :return: Gibt die dekodierten Wörter als Liste,
    die Decoder-Attention als Tensor zurück, die
    vorhergesagte Paraphrase als String und die
    Werte von BLEU und der Kosinus-Ähnlichkeit
    als Float zurück.
    """
```

```
def print_evaluation(sentence, output_sentence, cosine_similarity,
                     bleu, target_sentence=None):
    """
    Gibt die berechneten Scores, den
    Ausgangssatz, den Zielsatz und den
    vorhergesagten Satz der Evaluation
    in der Konsole aus.
    :param sentence: Ein String mit Wörtern, welche durch
    Leerzeichen getrennt sind.
    :param output_sentence: Vorhergesagte Paraphrase
    als String
    :param cosine_similarity: Berechnete Kosinus-Ähnlichkeit
    als Float
    :param bleu: Berechneter BLEU-Score als Float
    :param target_sentence: Optionaler Zielsatz zum Evaluieren
    von Paraphrasen im Trainingsprozesses.
    :return:
    """
```

```
def evaluate_textblock(original_text):
    """
    Evaluiert durch die Verwendung der Funktion
    evaluate(sentence) einen beliebigen Text, welcher als
    String übergeben wird und beliebig viele Sätze enthalten kann.
    Erzeugt eine Paraphrase für jeden Satz im übergebenen Text.
    :param original_text: Ein Eingabetext als String
    im UTF-8 Format
    :return: Gibt den Paraphrasierten Text als String
    und die Scores als Float zurück
    """
```

```
def evaluate_test_set():
    """
    Evaluiert durch die Verwendung der Funktion
    evaluate(sentence) alle Sätze aus der
    Textdatei der Testdaten. Die Funktion gibt
    die Scores, den Ausgangssatz und den
    vorhergesagten Satz in der Konsole aus und
    berechnet die Durchschnittswerde für die
    verwendeten Metriken.
    :return:
    """
```

# Glossar

| | |
|---|---|
| Domäne | Im Kontext dieser Arbeit bezeichnet eine Domäne ein spezifisches Fachgebiet. |
| Paraphrase | Eine Textvariante eines bestehenden Textes, welche den Kontext unverändert in anderen Worten wiedergibt. |
| Paraphrasen-Tupel / Paraphrasenpaar | Ein bereinigter Originaltext und eine dazugehörige Paraphrase dieses Textes. |
| Rohtext | Text, welcher noch vorverarbeitet werden muss, da er hinsichtlich des Formats und der Struktur noch nicht den Normen des Systems entspricht. |
| Subdomäne | Ein Teilbereich innerhalb einer Domäne. (Beispielsweise: $Regen \subseteq Wetter$) |
| Template | Im Kontext dieser Arbeit handelt es sich bei einem Template um eine strukturelle und inhaltliche Vorlage zum automatisierten Generieren von Text. |
| Textvarianz | Ein wesentliches Qualitätsmerkmal von Text, welches das Abweichungsausmaß zwischen zwei Texten beziffert. |

# Abbildungsverzeichnis

# Literaturverzeichnis

[Bah16] **Bahdanau 2016**
BAHDANAU, Dzmitry ; CHO, Kyunghyun ; BENGIO, Yoshua: Neural Machine Translation by Jointly Learning to Align and Translate. : Submitted on 1 Sep 2014 on https://arxiv.org. - arXiv:1409.0473v7

[Bar13] **Barancíkova 2013**
BARANCÍKOVA, P. : Lexical Paraphrasing for Improvement of MT Evaluation. : WDS'13 Proceedings of Contributed Papers, Part I, 7–11, 2013 – ISBN: 978-80-7378-250-4

[Bro18] **Brownlee 2018**
BROWNLEE, Jason: A Gentle Introduction to Exploding Gradients in Neural Networks. In: www.machinelearningmastery.com.
URL: https://machinelearningmastery.com/exploding-gradients-in-neural-networks/ (letzter Abruf am 12.12.2018)

[Bro17] **Brownlee 2017**
BROWNLEE, Jason: Teacher forcing for recurrent neural networks
In: www.machinelearningmastery.com.
URL: https://machinelearningmastery.com/teacher-forcing-for-recurrent-neural-networks/ (letzter Abruf am 14.02.2019)

[Bro16] **Brownlee 2016**
BROWNLEE, Jason: What is Deep Learning?
In: www. machinelearningmastery.com.
URL: https://machinelearningmastery.com/what-is-deep-learning/
(letzter Abruf am 11.12.2018)

[Cas16] **Castrounis 2016**
CASTROUNIS, Alex: Artificial Intelligence, Deep Learning, and Neural Networks, Explained.
In: www.kdnuggets.com.
URL: https://www.kdnuggets.com/2016/10/artificial-intelligence-deep-learning-neural-networks-explained.html
(letzter Abruf am 14.01.2019)

[Cha17] **Changhau 2017**
CHANGHAU, Isaac: Loss Functions in Neural Networks.
In: www.isaacchanghau.github.io.
URL: https://isaacchanghau.github.io/post/loss_functions/ (letzter Abruf am 14.02.2019)

[Cho14] **Cho 2014**
CHO, Kyunghyun ; MERRIENBOER, Bart ; GULCEHRE, Caglar ; BAHDANAU, Dzmitry ; BOUGARES, Fethi ; SCHWENK, Holger ; YOSHUA, Bengio: Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. : Submitted on 3 Jun 2014 on https://arxiv.org. - arXiv:1406.1078v3

[Com16] **Complx 2016**
COMPLX: Chatbots with Seq2Seq:
Learn to build a chatbot using TensorFlow.
In: www.complx.me.
URL: http://complx.me/2016-06-28-easy-seq2seq/
(letzter Abruf am 08.01.2019)

[Cro15] **Crocetti 2015**
CROCETTI, Giancarlo : Textual Spatial Cosine Similarity. : Submitted on 15 May 2015 on https://arxiv.org. - arXiv:1505.03934

[Don18] **Donges 2018**
DONGES, Niklas: Recurrent Neural Networks and LSTM.
In: www. towardsdatascience.com.
URL: https://towardsdatascience.com/recurrent-neural-networks-and-lstm-4b601dd822a5 (letzter Abruf am 10.12.2018)

[Gér17] **Géron 2017**
GÉRON, Aurélien : Hands-On Machine Learning with Scikit-Learn & Tensorflow. : O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472 – ISBN: 978-1-491-96229-9

[Gup18] **Guptra 2018**
GUPTA, Sanket: Overview of Text Similarity Metrics in Python.
In: www. towardsdatascience.com.
URL: https://towardsdatascience.com/overview-of-text-similarity-metrics-3397c4601f50 (letzter Abruf am 14.02.2019)

[Isi18] **Isichei 2018**
ISICHEI, Faith: BLEU in BuildAnalytics.
In: www.kantanmt.zendesk.com.
URL: https://kantanmt.zendesk.com/hc/en-us/articles/205355285-BLEU-in-BuildAnalytics (letzter Abruf am 14.02.2019)

[Kar18] **Karani 2018**
KARANI, Dhruvikl: Introduction to Word Embedding and Word2Vec
In: www. towardsdatascience.com.
URL: https://towardsdatascience.com/introduction-to-word-embedding-and-word2vec-652d0c2060fa (letzter Abruf am 14.02.2019)

[Khu17] **Khurana 2017**
KHURANA, Diksha; KOLI, Aditya ; KHATTER, Kiran ; SINGH, Sukhdev : Natural Language Processing: State of The Art, Current Trends and Challenges. : Submitted on 17 Aug 2017 on https://arxiv.org. - arXiv:1708.05148v1

[Kie05] **Kiesel 2005**
KIESEL, David : Ein kleiner Überblick über neuronale Netze. 2005 – Online verfügbar unter www.dkriesel.com/science/neural_networks

[Kos17] **Kostadinov 2017**
KOSTADINOV, Simeon: Understanding GRU networks.
In: www. towardsdatascience.com.
URL: https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be (letzter Abruf am 10.12.2018)

[Kum18] **Kumar 2018**
KUMAR, Chethan: NLP vs NLU vs NLG (Know what you are trying to achieve) NLP engine (Part-1). In: www.towardsdatascience.com.
URL: https://towardsdatascience.com/nlp-vs-nlu-vs-nlg-know-what-you-are-trying-to-achieve-nlp-engine-part-1-1487a2c8b696 (letzter Abruf am 10.12.2018)

[Luo15] **Luong 2015**
LUONG, Minh-Thang ; PHAM, Hieu ; MANNING, Christopher D. : Effective Approaches to Attention-based Neural Machine Translation. : Submitted on 17 Aug 2015 on https://arxiv.org. - arXiv:1508.04025

[Mey18] **Meyer 2018**
MEYER, Sean: How websites (actually) make money.
In: www.medium.com.
URL: https://medium.com/swlh/how-websites-actually-make-money-40f98897bd8d (letzter Abruf am 08.01.2019)

[Ngu18] **Nguyen 2018**
NGUYEN, Michael: Illustrated Guide to LSTMs and GRUs: A step by step explanation. In: www. towardsdatascience.com.
URL: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21 (letzter Abruf am 10.12.2018)

[Pap02] **Papineni 2002**
PAPINENI, Kishore ; ROUKOS, Salim ; WARD, Todd ; DATLA, Vivek ; WEI-JING, Zhu: BLEU: a Method for Automatic Evaluation of Machine Translation. : Submitted on July 2002 on https://aclanthology.info/papers/P02-1040/p02-1040 (letzter Abruf am 14.02.2019)

[Phr18] **Phrasetech 2018**
PHRASETECH: What's the difference between NLP, NLU and NLG?
In: www.phrasetech.com. URL: https://www.phrasetech.com/whats-difference-nlp-nlu-nlg/ (letzter Abruf am 10.12.2018)

[Pra16] **Prakash 2016**
PRAKASH, Aaditya ; HASAN, Sadid A. ; LEE, Kathy. ; DATLA, Vivek. ; QADIR, Ashequl. ; LIU, Joey. ; FARRI, Oladimeji: Neural Paraphrase Generation with Stacked Residual LSTM Networks. : Submitted on 10 Oct 2016 on https://arxiv.org. - arXiv:1610.03098v3

[Rob17] **Robertson 2017**
ROBERTSON, Sean: TRANSLATION WITH A SEQUENCE TO SEQUENCE NETWORK AND ATTENTION
In: www.pytorch.org.
URL: https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html (letzter Abruf am 14.02.2019)

[Rud16] **Ruder 2016**
RUDER, Sebastian: On word embeddings - Part 1
In: www. ruder.io.
URL: http://ruder.io/word-embeddings-1/ (letzter Abruf am 14.02.2019)

[Sch18] **Schmidhuber 2018**
SCHMIDHUBER, Jürgen : Deep Learning in Neural Networks: An Overview. : Submitted on 30 Apr 2014 on https://arxiv.org. - arXiv:1404.7828v4

[Shi18] **Shi 2018**
SHI, Zhan ; XINCHI, Chen ; XIPENG, Qiu ; XUANJING, Huang : Toward Diverse Text Generation with Inverse Reinforcement Learning. : Submitted on 30 Apr 2018 on https://arxiv.org. - arXiv:1804.11258v3

[Sky18] **Skymind 2018**
SKYMIND: A Beginner's Guide to Neural Networks and Deep Learning.
In: www. skymind.ai. URL: https://skymind.ai/wiki/neural-network
(letzter Abruf am 10.12.2018)

[Src1] **DWD 2019 - Daten API des deutschen Wetterdienstes**
https://opendata.dwd.de/weather/local_forecasts/mos/MOSMIX_S/all_stations/kml/
(Letzter Aufruf: 09.03.2019)

[Src2] **Vortrainierte Word2Vec Embeddings**
https://devmount.github.io/GermanWordEmbeddings/
(Letzter Aufruf: 12.02.2019)

[Src3] **Pytorch Seq2seq Tutorial**
https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html
(Letzter Aufruf: 16.02.2019)

[Sut14] **Sutskever 2014**
SUTSKEVER, Ilya ; VINYALS, Oiol ; L, Quoc V. : Sequence to Sequence Learning with Neural Networks. : Submitted on 10 Sep 2014 on https://arxiv.org. - arXiv:1409.3215v3

[Ujj16] **Ujjwalkarn 2016**
UJJWALKARN: A Quick Introduction to Neural Networks.
In: www. ujjwalkarn.me. URL: https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/ (letzter Abruf am 10.12.2018)

# Versicherung über Selbstständigkeit

*Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.*

*Hamburg, den* _____ _____