TECHNISCHE HOCHSCHULE NÜRNBERG
GEORG SIMON OHM

Faculty of Computer Science

IT-based Automatic Text Summarization with the use of Text Generation Methods

State of the art and design of a prototype

Bachelor Thesis in
Information Systems and Management

by

Tim Löhr

Student ID 3060802

First advisor:	Prof. Dr. Alfred Holl
Second advisor:	Prof. Dr. Florian Gallwitz

Hinweis: Diese Erklärung ist in alle Exemplare der Abschlussarbeit fest einzubinden. (Keine Spiralbindung)

TECHNISCHE HOCHSCHULE NÜRNBERG
GEORG SIMON OHM

Prüfungsrechtliche Erklärung der/des Studierenden

Angaben des bzw. der Studierenden:

Name: Vorname: Matrikel-Nr.:

Fakultät: Studiengang:

Semester:

Titel der Abschlussarbeit:

Ich versichere, dass ich die Arbeit selbständig verfasst, nicht anderweitig für Prüfungszwecke vorgelegt, alle benutzten Quellen und Hilfsmittel angegeben sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Ort, Datum, Unterschrift Studierende/Studierender

Erklärung zur Veröffentlichung der vorstehend bezeichneten Abschlussarbeit

Die Entscheidung über die vollständige oder auszugsweise Veröffentlichung der Abschlussarbeit liegt grundsätzlich erst einmal allein in der Zuständigkeit der/des studentischen Verfasserin/Verfassers. Nach dem Urheberrechtsgesetz (UrhG) erwirbt die Verfasserin/der Verfasser einer Abschlussarbeit mit Anfertigung ihrer/seiner Arbeit das alleinige Urheberrecht und grundsätzlich auch die hieraus resultierenden Nutzungsrechte wie z.B. Erstveröffentlichung (§ 12 UrhG), Verbreitung (§ 17 UrhG), Vervielfältigung (§ 16 UrhG), Online-Nutzung usw., also alle Rechte, die die nicht-kommerzielle oder kommerzielle Verwertung betreffen.

Die Hochschule und deren Beschäftigte werden Abschlussarbeiten oder Teile davon nicht ohne Zustimmung der/des studentischen Verfasserin/Verfassers veröffentlichen, insbesondere nicht öffentlich zugänglich in die Bibliothek der Hochschule einstellen.

Hiermit ☐ genehmige ich, wenn und soweit keine entgegenstehenden Vereinbarungen mit Dritten getroffen worden sind,

☐ genehmige ich nicht,

dass die oben genannte Abschlussarbeit durch die Technische Hochschule Nürnberg Georg Simon Ohm, ggf. nach Ablauf einer mittels eines auf der Abschlussarbeit aufgebrachten Sperrvermerks kenntlich gemachten Sperrfrist

von Jahren (0 - 5 Jahren ab Datum der Abgabe der Arbeit),

der Öffentlichkeit zugänglich gemacht wird. Im Falle der Genehmigung erfolgt diese unwiderruflich; hierzu wird der Abschlussarbeit ein Exemplar im digitalisierten PDF-Format auf einem Datenträger beigefügt. Bestimmungen der jeweils geltenden Studien- und Prüfungsordnung über Art und Umfang der im Rahmen der Arbeit abzugebenden Exemplare und Materialien werden hierdurch nicht berührt.

Ort, Datum, Unterschrift Studierende/Studierender

Preface I

The following thesis was created during the seventh and last semester at the Georg Simon Ohm University of Applied Science. Within the last three semesters, I realized that my major interest among all IT related topics is artificial intelligence.

My personal interest started basically with a group IT-project, in which my team and I programmed an autonomously driving remote control car with a deep neural network together with a Raspberry Pi 3B+. From this first project on, I selected all my further elective courses to be related to machine learning or data science in any possible way. I wanted to increase my knowledge further, so I searched for a website that provides courses related to AI. I found *www.udacity.com*, which offers courses in cooperation with top IT companies, such as Google, Airbnb, or Microsoft. Out of curiosity, I bought the course *Natural Language Processing*. After successfully finishing it, I was encouraged to write my bachelor thesis in a *Natural Language Processing* related topic. Together with my professor *Prof. Dr. Alfred Holl*, I worked out a structured methodological table for the entire structure of this paper. Even though Natural Language Processing is just a subfield of machine learning, the current state-of-the-art research is far beyond what I can research within a bachelor thesis. I decided to write my thesis about the subfield *textgeneration* within NLP. My state-of-the-art research includes all *hot topics* within NLP, and my prototype focuses only on the text generation part, to dive deeper into what NLP and especially text generation can accomplish in the year 2020.

Preface II

For my research, I encountered a lot of old and recently published papers, mostly from *https://arxiv.org/*. To read through the papers requires a lot of prior knowledge, especially in mathematics, which I learned during my semester in Hong Kong at the City University of Hong Kong. To fully understand the mathematics given in this thesis, enhanced knowledge of calculus and linear algebra is required. Even if this is not the case, I will describe the process in such a way that it can be comprehended without looking at the maths.
Machine Learning and, more specifically, NLP is not an intuitive study. I provided for the matrix notations the common terminologies originated from top researchers and tried to make the entry into this field as smooth as possible if the reader has no prior knowledge about this topic. During the five-month development process of the bachelor thesis, I gained much knowledge. I recognized that NLP is a huge topic, constantly under research. To keep up to date with the latest publications requires much effort.
To give a full state-of-the-art review about *all* NLP related disciplines is not possible within this thesis. For this reason, I focus entirely on the development of the *Neural Text Generation* (NTP), which includes more fields than the reader might imagine.

	Titel / Kapitel		Untertitel / Unterkapitel	——Woher?—— Wissensinput	——Woher?—— Frageinput	——Wie?—— Methode	——Was?—— Zielbeschreibung
0	**IT-basierte Textgenerierung mit Hilfe von NLP-Methoden** **State of the Art & Entwurf eines Prototypen**			Allgemeingültig: • Fachbücher, Bücher • HongKong, TH-OHM • Online Kurse	1. Was ist der State of Art von NLP - Systemen. 2. In welcher Qualität kann ich den Textgenerierungs- Prototypen selbst programmieren und welche Güte hat dieser?	1. Darstellung des State of the Art der NLP-Systeme. 2. Studium der relevanten Aspekte des NLP und Programmierung eines IT-basierten Textgenerierungs-Prototypen.	1. State of the Art fachlich herausarbeiten. 2. Einen Prototypischen Algorithmus programmieren, der zu einem gegeben Input z.B. ein Buch immer wieder neue kreative Fortsetzungen generiert.
1	**Einleitung**	**1.1**	**Fallbeispiel eines aktuellen NLP-Systems**	• [0.1] • Wissenschaftliches Schreiben und	1. Was sind aktuelle, nützliche Einsatzgebiete von NLP-Textverarbeitungs-Systemen? 2. Was ist der Nutzen meines NLP-Prototypen im Bereich der Textverarbeitung?	1. Recherche über die aktuellen und geplanten NLP-Systeme, im Bereich der Textverarbeitung. 2. Vorstellung meines Beitrags zu NLP-Systemen mithilfe meines Prototyps.	1. Antwort auf die Frage, warum meine Bachelorarbeit sinnvoll ist und welche Motivation ich habe zur Bearbeitung 2. Erläuterung durch einen interessanten leichten Einstieg.
2	**State of the Art**	**2.1**	**Relevante Aspekte der Mathematik**	[1.1]	Welches mathematische „know-how“ ist notwendig, um NLP-Systeme für Textverarbeitung und meinen Prototypen technisch verstehen zu können?	Recherche nach den relevanten Aspekten der Mathematik für dieses Thema.	Beschreibung der anwendungsbezogenen mathematischen Modelle für diesen Themenkomplex anhand von Formeln und Erklärungen.
		2.2	**Geschichte des NLP**	• [0] • [0.1]	1. Seit wann wird an NLP-Systemen geforscht? 2. Ab welchem Punkt konnte man effektiven Nutzen aus diesen Systemen ziehen?	1. Literaturrecherche über die Geschichte des NLP (40 Jahre). 2. Literaturrecherche über die ersten Einsätze der NLP-Systeme.	1. Darstellung der Geschichte des NLP in Form einer zeitlichen Abfolge. 2. Nutzen der ersten NLP-Prototypen oder Technologien die im Einsatz waren.
		2.3	**Aktuelle Trends der Technologie**	• [0] • [0.1] • [2.2] • Fallbeispiele	1. Was sind aktuelle NLP-Systeme imstande zu leisten? 2. Wo sind die Einsatzgebiete?	1. Literaturrecherche über aktuelle Trends (+ - 5 Jahre). 2. Recherche von aktuelle Papern und Veröffentlichungen.	1. Darstellung der aktuellen Technologien. 2. Blick in die kurzfristige Zukunft anhand von aktuellen Fallbeispielen und Forschungsergebnissen.
3	**Prototyp**	**3.1**	**Zielsetzung / Anforderungen**	• [0] • [1] • [2]	1. Was soll mein Prototyp mit gegebenen Mitteln leisten können? 2. Welcher Output ist im besten Fall zu erwarten?	1. Requirements Engineering. 2. Klassifizierung und Analyse möglicher Ergebnisse, z.B. ob der Output grammatikalisch korrekt ist.	1. Erläuterung des Umfangs meines Prototyps. 2. Sammlung und Klassifizierung der Anforderungen an den Algorithmus und dessen Output.
		3.2	**Fachkonzept**	[3]	1. Wie ist mein Prototyp strukturiert? 2. Welche Algorithmen verwende ich? 3. Welche Prozesse durchlaufen die zu verarbeitenden Daten? 4. Wie werden die Daten verarbeitet?	1. Erstellen eines Fachkonzepts 2. Algorithmus modellieren 3. Prozessmodellierung 4. Datenflussmodellierung und, oder Datenmodellierung	1. Fachkonzept fertig erstellt. 2. Der Prototyp wird ohne IT Bezug anhand von verschiedenen Teilmodellen modelliert. 3. Die einzelnen Prozesse werden ohne konkreten Implememtierungs-Vorschlag modelliert. 4. Datenverarbeitung visualisiert
		3.3	**Implementierung**	[3.3]	1. Welche Technologien verwende ich für meinen Prototypen: - „Welche Python Bibliotheken und IDE?“ - „Welche HW & SW-Anforderungen gibt es?“ 2. Welche Probleme traten bei der Programmierung auf?	1. Software-Abhängigkeits-Portfolio erstellen - Vergleich geeigneter Programmiersprachen - Recherche der erforderlichen Bibliotheken - Recherche der erforderlichen Hardware, Software und Auswahl 2. Software entwickeln - Fehler reporten an Hersteller, Bib, etc.	1. Erstellung eines IT-Konzepts in Form einer Beschreibung der notwendigen technischen Mittel anhand von Teilmodellen 2. Problemstellungen erklären und das Auftreten eines Problems „reverse Engineeren“
		3.4	**Evaluation**	[3.4]	1. Wie ist der Output des Prototyps zu bewerten? 2. Wie bewertet man die Qualität des Outputs? 3. Was kann verbessert werden?	1. Soll-Ist-Vergleich der Anforderungen mit dem Output des Prototypen. 2. Vergleich mit verwandten Arbeiten. 3. Recherche über potentielle Verbesserungen des Algorithmus.	1. Evaluation und Analyse des Ergebnisses anhand von grammatikalischer Richtigkeit und Sinn. 2. Bessere Ergebnisse mit meinen vergleichen. 3. Optimierungsmöglichkeiten für meinen Prototypen evaluieren.
4	**Generierung von übertragbarem Wissen**			[0] bis [3]	Um welche Elemente könnte mein Projekt modular Erweitert werden um ein Anderes oder Besseres Ergebnis zu erzeugen und welchen Einfluss könnte es auf die Forschung haben?	Verallgemeinerung aus den bisher erarbeiteten Ergebnissen.	Einordnung der Evaluationsergebnisse in einen gesellschaftlichen Kontext.

Abstract

– At the end , finally finished !!! –

Contents

Chapter 1

Introduction

The 21st century is flushed with a massive amount of texts and documents. Every day there are new articles, news, documentation and reports packed full of information. For this reason, a new discipline arose out of this. Knowledge is nowadays accessible everywhere and immediately, but consumption takes way too much time. Websites like *https://www.blinkist.com/de* provide their costumer's text summarizations of different kinds of books readable in 15-30 minutes. This is an exciting way to save time, but still, this summarization is done by hand. Artificial Intelligence researchers continuously provide knowledge to the public to summarize text with computer algorithms. The first approaches of automatic text summarization were grammatically wrong and reading grammatically broken summarizations is tiring for the most people. Deep Learning changed the game entirely, because algorithms are now feasible enough to summarize texts as good as humans do.

1.1 Structure of the thesis

The aim of my thesis is to survey the current state of the art in text generation, especially on the focus of text summarization. The development into the state of the art neural text summarization had a huge impact on the readability for the human. For readers who are not familiar with machine learning in general, I will provide a zoom-in introduction from artificial intelligence in general into the tiny sub field text summarization. My approach is feed-forward from the definition of machine learning, deeper into the natural language processing field, further into the text generation field and within that, I focus on the text summarization part in chapter 1 - Introduction. New research and state of the art results in some natural language processing fields often lead to improvements across other related disciplines in machine learning and natural language processing, because algorithms are sometimes usable vice-versa. For this reason, I provide the most crucial text generation historical achievements in combination with the latest text summarization results, because both topics intersect in many aspects. The crucial concept of historical and modern approaches to summarize and generate text are introduced in chapter 2 - An evolutionary view on the State of the Art. To illustrate the basic workflow of a text summarizing system, I programmed a prototype.

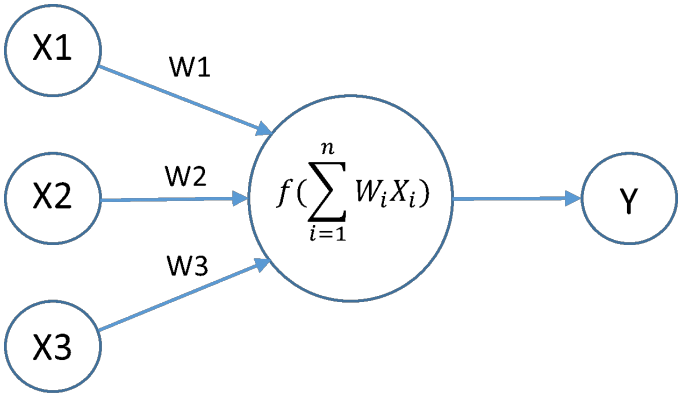

Figure 1.1: A simple Neuron with 3 inputs and 1 output [Sing 17]

The concept, development and evaluation of this summarizer are located in chapter 4 - Prototype, but it requires prior knowledge to fully understand the mechanism from the input to the output. Finally in the last chapter I will discuss further improvements for my prototype and a brief discussing of future of text generation.

1.2 Machine Learning

In the last decade, Machine Learning (ML) is increasingly finding its way into businesses and society. Many websites and businesses use Machine Learning techniques to improve user and customer experience. The phrase *Machine Learning* was initially introduced in 1952 by Arthur Samuel. He developed a computer program for playing the game checkers in the 1950s. Samuel's model was based on a model of brain cell interaction by Donald Hebb from his book called *The Organization of Behavior* published in 1949. Hebb's book introduces theories on neuron excitement and neural communication. Figure 1.1 illustrates a mathematical approximation of the human's brain cell in the form of a *artifical* neuron. Nowadays, this brain-neuron based model is mostly declared to be not close enough to reality [Andrew Ng, deeplearning.ai], because the structure of a brain's neuron is far more complicated than the illustration in figure 1.1 suggests. Nevertheless, it provides an excellent entry point for this research field in my opinion.

The roots of Neural Networks (NN) lie down almost 80 years ago in 1943 when **McCulloch-Pitts** [McCu 43] compared for the first time neural networks with the structure of the human brain. The range in which Neural Networks (in the year 2020) apply to modern technologies is wide. Some disciplines have only been created due to the invention of Neural Networks, because they solve existing and new problems more effectively and efficiently than previously used algorithms. Many frequently held conferences around the globe prove continuous evidence of the successes of Neural Networks. Among those various disciplines counts for

example *Pattern recognition* with Convolutional Neural Networks (CNN) [Yann 98]. Convolutional Neural Networks are one of the many special building blocks of the neural network. Every building block aims to solve a different task. For example, Pattern recognition uses different layers (building blocks) in its neural network than text summarization, because the input for Pattern recognition neural networks is often a picture consisting of e.g., 32x32 pixels. In contrast, the input for the text summarization is e.g., a 1000 word long text.

A widely known entry challenge into pattern recognition is the *CIFAR-10* dataset [Kriz]. It consists of 50.000 images divided into ten classes of different objects and animals like cats and cars (5000 images of cats, 5000 images or cars, ...). Classification algorithms try now to predict a class for the input image as precisely as possible. Many amateurs [Löh 19] and experts annually attempt to show their latest results in beating the former best accuracy.

Natural Language Processing is one of the various sub-fields of Machine Learning. Strictly speaking, it is a multidisciplinary field consisting of Artificial Intelligence (AI) and computational linguistics. Natural Language Processing is dedicated to understand and process the interactions between human (natural) language and computers. Natural Language Processing is a broad term and can be applied to many different tasks, such as:

- **Sentiment Analysis**, e.g. Google Reviews on Restaurants
- **Machine Translation**, e.g. Google Translator
- **Speech Recognition**, e.g. Siri from Apples IPhone
- **Text Generation (Neural Text Generation [*NTG*])**, e.g. Text Summarization
- **Chat Bots**, e.g. Shopping Websites

Deep Learning is not an absolute definition. Many top researchers define it in a very different way. In general, Deep Learning allows building more complex neural networks, which are capable of detecting better and more correlation in data. Figure 1.2 shows the zoom-in from AI to Deep Learning. Therefore Deep Learning can be seen as a method in Machine Learning, not to mix up with Natural Language processing, which can make use of Deep Learning techniques, but it is not required to use it.

All of these tasks require many steps to function correctly. In the broadest sense, there is always an Input and an Output, which are shown in Table 1.1.

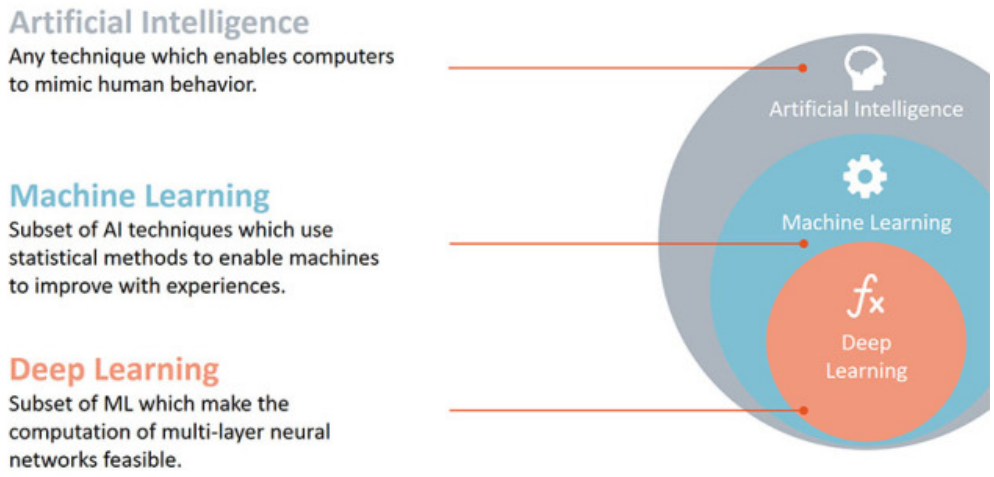

Figure 1.2: Zoom into Artificial Intelligence from *https://rapidminer.com/blog/artificial-intelligence-machine-learning-deep-learning/*

Example components of Input - Output systems			
	Speech	Text	Images
Input Analysis	Speech Recognition	Text Recognition	Image Recognition
Output Synthesis	Generation of Speech	Generation of Text	Generation of Images
Processing method	NLP method	NLP method	CNN Building Blocks

Table 1.1: A closer look into Input Output systems with the focus on Text Generation

Examples of Natural Language Processing systems			
	Speech	Text	Text
Input Analysis	Siri listens	Read in document	Read in document
Output Synthesis	Siri answers	Generate Summary	Generate sentiment

Table 1.2: Examples for three different NLP tasks

It shows that Text Generation is the **output part** of a **Natural Language Processing** model. Data is collected through various sources, e.g. images, videos or speech, then it is

further processed and generates the desired output. Useful examples are shown in Table 1.2.

For this Bachelor thesis, the focus is on the output part of a Natural Language Processing system. More specifically, the text summarization which inputs text as shown in Table 1.1 and 1.2 and outputs the summary is what I treat in my work. Text generation is therefore generally the output-synthesis part of an input-output NLP system.

However, what defines a summarization and what defines a good summarization? The literature points out multiple different definitions. One definition proposes that the summary of a document is the process of distilling the essential information from a source (or sources) to produce an abridged version for a particular user (or users) and task (or tasks) [Mani 99b]. Its objective is to give information and provide classified access to the source documents. Summarization is an automatic task when it is generated by software or an algorithm.

Another term for Text Generation is *Language Modelling*, because text generators use the words of a language and grammar as input for the model. In the past five years, primarily two approaches were used for modeling a Natural Language Processing system, namely the **rule-based** system and the **template-based** system (Figure 1.3) [Xie 17]. Today neural end-to-end systems are *state-of-the-art* [Jeka 17]. These systems offer more flexibility and scale with proportionately better results, and less data is required because of the increased complexity. These systems are called neural, because they make use of **Deep Learning** Neural Networks. A significant disadvantage is that the necessary computing power has increased exponentially. This computing power requires a strong server and most language models are trained on a very large scale. This can most commonly not be achieved by a single person at home, except if one rents for example a strong server from the *Amazon Web Services*. The computational requirement rose in such a way, because of the more profound and deeper neural networks nowadays. However, this leads to a complex problem because it becomes more and more challenging to understand the decisions of the neural network. The neural network is still, to a large extent, a *black box*. Nevertheless, especially in NLP, it gives surprisingly good results. The neural network models for text processing are difficult to understand, so nowadays, compromises between rule-based systems still have to be made, and hybrid systems are most commonly in use.

When Neural end-to-end systems are used, Text Generation is often referred to as Neural Text Generation (NTG). More examples for Neural Text Generators as output synthetical component are:

- Speech recording and conversion to text
- Conversation systems e.g. chatbots
- Neural Text summary

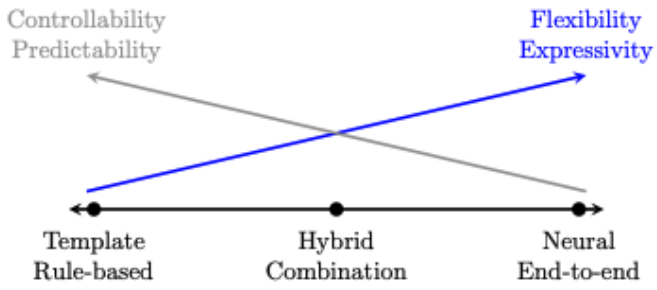

Figure 1.3: Rule-Based vs. Neural-Text-Generations System [Xie 17], Page 4

- Caption generation of Images

In order to train language models properly, Deep Learning (DL) algorithms teach the model the probabilities of occurring words with respect to the preceding words. There are several approaches to achieve this goal. Language models can be trained on the level of words, whole sentences, or even whole paragraphs. The granularity in which the training takes place is called *n-grams*, where n represents the number of preceding words.

1.3 Case study of an Automatic Text Summarization System (ATS)

As a human, creating a good summary of a text requires that the person understood the text well. The text needs to be understood so well, that the person can summarize the essence of the text in such a way, that it shortens the original document to a minimum down without cutting out important information. However, after a while, a person is most likely to summarize the same text differently than one month ago. Due to this circumstance, the summarization task tends to be challenging to automate. Depending on what kind of summary is needed, the texts must be processed differently into different fragments with multiple relevances for each fragment. A crucial role is also the coherence of a text. Different applications of text summarizations are:

- Web Page Summarization
- Reports or Meetings
- Opinion Summarizations
- Scientific Research Papers
- News Headlines

Input: Article 1st sentence	Model-written headline
metro-goldwyn-mayer reported a third-quarter net loss of dlrs 16 million due mainly to the effect of accounting rules adopted this year	mgm reports 16 million net loss on higher revenue
starting from july 1, the island province of hainan in southern china will implement strict market access control on all incoming livestock and animal products to prevent the possible spread of epidemic diseases	hainan to curb spread of diseases
australian wine exports hit a record 52.1 million liters worth 260 million dollars (143 million us) in september, the government statistics office reported on monday	australian wine exports hit record high in september

Table 1.3: The first column shows the first sentence of a news article which is the model input, and the second column shows what headline the model has written [Scie 15].

Google News Headline Summarization

Google has its own news section on this link *https://news.google.com*. Google News automatically generates the news headlines for multi-language news articles [Scie 15]. Google proposes that for people to digest a large amount of daily information better, they created the long-term goal at the *Google Brain* department to summarize news articles and their headlines as pleasant as possible. The search engine Google is known for its accurate and many search results when using this engine. Google does the same for Google News. Google scrapes news articles all over the world, automatically summarizes it, and out of that, it generates the headline of the summarized news article out of it [Scie 15]. For achieving their state of the art results, Google makes use of a Deep Learning technique called *sequence-to-sequence* learning, which will be explained in Section 2.2.3. Table 1.3 shows an example for generated headlines of a summarization. Due to the structure of a news article, a good headline summary requires most likely only the first few sentences of an article.

Many examples from this thesis will be conducted from the perspective of a news article to keep it uniform, whether a long text will be summarized to a shorter text or only one or two sentences for a headline will be extracted and summarized from an article.

Chapter 2

An Evolutionary View on the State of the Art

The goal of this chapter is to survey the development of the text generation from the old days until 2020.

As conducted from the Introduction Chapter 1, Text Generation is the generic term for the output part of an automatic text summarizer. The research on Neural Text Generation and other fields had a significant impact on the development of automatic text summarizers. In this chapter, I begin with the definition of a text generator in general and its historical development. I state the most important steps from a basic text generator to a neural text generator. In the following, I focus on the text summarizer and its historical evolution with the impacts of the neural text generators. There are several human-like summarizing state of the art technologies nowadays for the automatic text summarizers, but they are developed under a large scale data set and computational high demanding power. This chapter is therefor the background knowledge for my prototype, but it also provides a peek into the latest state of the art technology which cannot be applied at home with a low power computer, but is only possible for big companies like Google.

2.1 Text Generation Concepts

Text Generation, Language Modeling or Natural Language Generation are different words with basically the same meaning, but I will keep the denotation of text generation throughout this thesis. A widely-cited survey from Reiter and Dale 1997 (Page 57-87) [Reit 97] characterizes text generation as 'the sub-field of Artificial Intelligence and computational linguistics that is concerned with the construction of computer systems than can produce readable texts in English or other human languages from some underlying non-linguistic representation of information' [Reit 97]. This definition implies rather a data-to-text approach instead of the text-to-text approach from Table 1.1, but in 1997 the rule-based approach dominated the neural end-to-end (Neural Text Generation) methods (Figure 1.3). For that reason, in 2003 Evans declares Text Generation as quite difficult to define [Evan 02] (Page 144-151). Most researchers agree on the text as the output synthesis part of the input-output

system (e.g. Text Summarization or Image caption generation [Mitc 12]), whereas the input part cannot be as easily distinguished [McDo 93] (Page 191-197).

2.1.1 Text Generation Tasks

For the Text Generation input-output system, the system can be divided into six sub-problems [Reit 97]. The following bullet points contain the six most crucial steps:

- **Content determination**: Deciding which information to include in the text under construction
- **Text structuring**: Determining in which order information will be presented in the text
- **Sentence aggregation**: Deciding which information to present in individual sentences
- **Lexicalisation**: Finding the right words and phrases to express information
- **Referring expression generation**: Selecting the words and phrases to identify domain objects
- **Linguistic realization**: Combining all words and phrases into well-formed sentences

These six tasks can be though of an early decision processes. They suggest both a chronological order in which the tasks need to be solved, as well as an distinction between strategy and tactics. This distinction goes back to Thompson H. in 1977, where he first declared this two parts [Thom 77]. Still, when it comes to modern neural state of the art Text Generation, the steps intersect in some ways. In the following comes a brief introduction to each of the steps. For the headline example is no aggregation necessary.

2.1.1.1 Content Determination

The first step is to determine which content should be present in the generated output text. Usually there is more information stored in the input than in the output. For this reason a certain *choice* must be undertaken for the content. As mention in the case study (Section 1.3, the headline can be summarized very precisely given only the first few sentences of the news. In this special case the determined content could be the first three sentences. For shorten a longer document into a summary, the key points need to be abstraced into a collection of preverbal messages and semantic representations of information, often expressed in the form of a logical or database like style [Gatt 18]. This means basically to group semtantical similar words and phrases together, to remove redundancies. This step followed for the

most time a rule-based approach, but in recent years researchers developed a data-driven approach (more in Section 2.1.2). For example, Barzilay and Lee (2004) developed a method to determine the content through Hidden Markov Models (HMM) [Barz 04] (Pages 113-120). Hidden Markow Models are stochastical models named after the russian mathematician A. A. Markow. They chain up different states of a system, in our case different topics of one or many news articles. This topics automatically will be clustered together as sentences based on the natural language semantical meaning [Gatt 18].

2.1.1.2 Text Structuring

After successfully deciding which contents will be used in the generated text, the structure or order of this fragments need to be determind. Given the example article from Table 1.3:

> Australian wine exports hit a record 52.1 million liters worth 260 million dollars (143 million us) in september, the government statistics office reported on monday

A good news headline should give all necessary information for the reader, namely:

- Where did it happen? -> *Australia*
- What happened? -> *Wine exports, record high*
- Who did something? -> *Australia*
- When did it happen? -> *September*

For our example the content was already predefined in the first step, now the important words and sentences will be reorder based on this four questions. Generalization approaches for the ordering task have already been proposed. Lapatas approach [Lapa 06] (Page 471-484) tries to find an optimal ordering of *information-bearing-items*. This method can even be applied to multi document input, which is more difficult to solve than single document inputs (explained in Section **??**).

2.1.1.3 Sentence Aggregation

By combining separate sentences with similar meaning into one, the generated text becomes potentially more fluid and enhances the readability [Dali 99] (Pages 383-414) [Chen 00] (Pages 183-193). For example, an aggregation makes sense for a football games and its results published in the Google News. Google could web scrape the live tickers of goals and after collecting all the data a possible result would be:

(1) Mario Götze scored after 19 minutes and 23 seconds
(2) Mario Götze scored after 20 minutes and 30 seconds
(3) Mario Götze scored after 60 minutes and 11 seconds

This is obviously not redundant, because it contains new information in every sentence, but for summarizing it, the sentences can be aggregates into:

(4) Mario Götze scored 3 times within 51 minutes

Aggregation is not an easy task, because it is not intuitive for an algorithm to detect semantic similarities and at the same time new information in that. Furthermore it depends highly on the to achieving output which kind of aggregation the text should undergo. A general approach was proposed by White and Howcraft (2015). They designed an algorithm to detect parallel verb phrases (*scored after*) in multiple (three) sentences and elide the subject and the verb in the generated sentence [Whit 15] (Pages 28-37).

2.1.1.4 Lexicalisation

After the sentences have been aggregated and finalized, the next step is lexicalisation, which converts the sentences into natural language. A single event can be expressed by natural language in multiple ways. For example the scoring event from the last section could be expressed as *scored three goals* or *goaled for three times*. The complexity for the lexicalisation step correlates with the amount of alternative sentences available. Furthermore it is important if there summary is limited with an amount of variation [M Th 01] (Pages 47-86). Whether or not the text shall be processed with lexical variation in its generated sentences or not depends on the application field. In needs to be decided in advance. For example the soccer game is more likely to be converted into a different styles than a weather forecast. Another important difficulty is to design the way on how the lexicalisation cares about gradable properties. For example if the liveticker was:

(1) Mario Götze scored fast after 3 minutes and 23 seconds

Then the systems needs to know whether the football player scored fast in a way that it is an early stage of the game, or he ran in such a fast way and scored with the pace. Humans tend to perceive different, as Power and Williams (2014) pointed out in an evaluation. A timestamp expression of *00:00* can be perceived as *midnight*, *late evening* or simply even *evening* for some people [Powe 14] (Pages 113-134).

2.1.1.5 Referring Expression

Referring Expression Generation is highly characterized by Dale and Reiter in 1997. They came up with the idea to identify words and phrases as domain entities. Nowadays this is also known as *Named Entity Recognition*. This step shows some similarity to lexicalisation, but Dale and Reiter pointed out that expression referring is *a discrimination task, where the system needs to communicate sufficient information to distinguish one domain entity from other domain entities* [Reit 00]. From the previous example, *Mario Götze* can be denoted with his name, another way would be calling him *football professional* or *the athlete*. Many factors play a role in how to determine which expressions and factors play a role in a particular context. Referring expression generation can basically be broken down into two steps. The first step is to decide the shape of referring expression. What type of reference should be used (e.g., a proper name or described with his/her job) [Cao 19]. The second is to determine the content of the referring expression (e.g., Mario Götze or the athlete) [Cao 19]. Rule-based approaches as well as the state of the art Machine Learning approaches have been proposed to solve this task [Reit 00].

The usual limitation of previous referring expression generation systems is that they are not able to generate referring expressions for new, unseen entities [Anja 10] (Pages 294-327). With the use of modern Machine Learning approaches, this limitation has overcome. Many tools, for example the *Natural Language Processing Toolkit NLTK* allow the easy transformation from the lexicalized sentence into its named entities. The sentence *Mario Götze scored fast after 3 minutes and 23 seconds* will be transformed into:

(1) [('Mario', 'NNP'),
(2) ('Götze', 'NNP'),
(3) ('scored', 'VBD'),
(4) ('fast', 'RB'),
(5) ('after', 'IN'),
(6) ('3', 'CD'),
(7) ('minutes', 'NNS'),
(8) ('and', 'CC'),
(9) ('23', 'CD'),
(10) (Seconds', 'NNS')]

- **NNP** = noun, proper, singular (1,2)
- **VBD** = verb, past tense (3)
- **RB** = adverb (4)
- **IN** = preposition or conjunction (5)
- **CD** = numeral, cardinal (6, 9)
- **NNS** = noun, common, plural (7, 10)
- **CC** = conjunction, coordinating (8)

2.1.1.6 Linguistic Realization

Finally after detecting all relevant words and structures of the input text, it only needs to be combined into a well-formed sentence. Usually referred to as linguistic realisation, this task involves ordering constituents of a sentence, as well as generating the correct morphological shapes (e.g verb conjugations) [Gatt 18] (Pages 18-20). The special task in this step is whether the generated output needs to make use of words not present in the given text. In the case of text summarization, this is often referred to as an *abstractive* or *extractive* approach (shown in Section 2.3.3). This task can be thought of an non-isomorphic (not reservable, because the same word can have different named entities dependent on the sentence) structure [Ball 15] (Pages 387-397). The three most common approaches for the realization are:

- Human-crafted templates
- Human-crafted grammar-based systems
- Statistical approaches

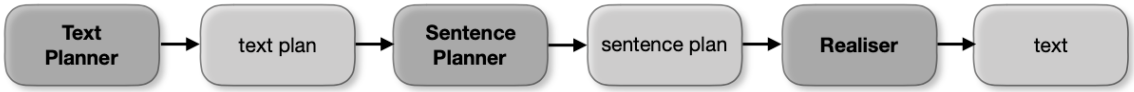

Figure 2.1: Classical 3-stage Text Generation architecture, after Reiter and Dale (2000) [Reit 00]

The most modern and widely used way is the statistical approach, but within that, there are several different methods to make us of the statistics. Just to give an example, Bohnet et al. (2010) [Bohn 10] (Pages 98-106) describe a realizer with underspecified dependency structures as input, Support Vector Machine (SVM) based environment. The classifiers are organised in a cascade to decode semantic input into the corresponding syntactic features. A Support Vector Machine is an algorithm to classify input whether it belongs to a specific topic or e.g. named entity from the last section or not. This can also be referred to as being a **Deep Learning** approach and it was applied on a common metric for measuring the accuracy of a generated e.g. text summarization called BLEU. This metric will be explained in Section 2.3.4. Furthermore, the more decision the statistical generating system makes and the more complex it becomes, the more abstract will the output be [**?**] (Page 21). This paves the way for an stochastical end-to-end system, like Konstas and Lapata showed in 2013 [Kons 13]. This presents a step words the automated text summarizations.

2.1.2 Architectures and Approaches

After the overview of the six main tasks for Text Generation systems, this section focuses on the way those tasks can be organized together. There are three main approaches for the Text Generation architecture shown in the dark boxes in Figure 2.1. The light boxes illustrate the outputs from the main stages.

Since the design of the modules in Figure 2.1, a lot has changed. The modular view is challenged by the planning-based and data-driven approach, because the modular view is not flexible enough for modern requirements, still it provides a good sequence structure and the original idea remains until now. The following three approaches will be explained in more detail in this section:

- Rule-based, modular approaches
- Planning-based approaches
- Data-driven approaches

2.1.2.1 Rule-based approach

The rule-based, or modular approach shown in Figure 2.1, is a classical approach from the early Artificial Intelligence research. It is designed to show a clear division between the sub-tasks, but with sometimes big variations among them. The three-stage architecture was originally called *consensius pipeline*, because the it is in the design of a pipeline and it was the de-facto standard in the year 2000 [Reit 00]. This pipeline share many similarities with the state of the art architecture used for text summarization in the year 2001 introduced by Mani et. al. [Mani 01]. It can be broken down into the following steps [Gatt 18] (Page 23):

- **Analysis** of the source text (single or multi document). This first stage includes *Text Planning*, which shares similar aspects with the Text Planner from Figure 2.1. One of the tasks for this step is *Content Determination*.

- **Transformation** of the selected input. This phase includes processing steps like *Text Structuring* and *Text Aggregation* on the selected text. It is especially important when it comes to abstractive text summarization (Section 2.3.3). This phase shares a number of similarities to the *Sentence Planner* from Figure 2.1.

- **Synthesis** produces the summary of the input based on the transformed selected input. The higher the abstraction level of the output should be, the more important is this phase. It can be seen as the *Realiser* with the methods of the *Linguistic Realiser* from the previous section.

The strict breakdown into clear stages (modules) comes with the cost of decreased flexibility. There is not always a rule for each task and the state of the art results are achieved by abstractive based methods for Text Generation. Those alternative approaches with a better abstraction level come on the other hand with the cost of blurred boundaries in the single stages. The basic idea is to create hand-crafted templates for all possible circumstances. To keep up with the football example, a template could look like this:

```
goals(
        player-name = 'Mario Götze',
        minute = 19,
        seconds = 23,
        player-number = 10
)

foul(
        by-player = 'Mario Götze',
        to-player = 'Thomas Müller',
        minute = '20',
        second = '12'
)
```

Even for a single football game it is obviously not possible to create a template with all possibilities. There may be some cases in which the possible combination of semantic units is not as big as in a single football game. In this cases a rule-based template could make sense, but for the most modern use-cases this approach is obsolete.

Even in the rule-based architectures have been many developments proposed, but for the sake of simplicity I want to discuss the modern approaches in more details than this old ones.

2.1.2.2 Planning-based approach

In the Artificial Intelligence field, the planning problem can be described as the process of detecting a sequence of one or more actions to satisfy an to achieving goal [Gatt 18] (Page 25). The classical planning-based approached was introduced by Fikes and Nilsson back in 1971 [Fike 71] (Pages 189-208). The idea was to to store actions into tuples containing of the preconditions and effects of the action respectively. In this way, planning-based means to regard language as an action [Clar 96].

Basically no restriction prevents the actions from choosing a type that can be inserted into a plan, plan-based approaches cut across the edges of many Natural Text Generation tasks that are normally strictly stacked in the classical pipeline architecture (Figure 2.1) [Gatt 18]. This means that the plan-based approach does not rely on the pipeline architecture, but steps are whirled together and follow different sequences than usually. This allows the input to be more flexibly processed. The most modern way for an planning-approach is the *stochastical planning using Reinforcement Learning*. Reinforcement Learning means, that the algorithm has an implemented reward and punishment variable, which allows the algorithm to notice

by itself when a certain action will be rewarded or punished. The reward and punishment need to manually configured. The Text Generation process could be modeled by an Long Short Term Memory (explained in Section 2.2.2. The time transitions t and the following $t+1$, ... , are associated with a reinforcement signal, via the reward or punishement function to adjust the behaviour of the wanted output.

Rieser et al. (2011) pointed out that this approach is effective in optimising information presentation when generating restaurant recommendations [Ries 11]. Janarthanam and Lemon (2014) applied this method to improve the choice of information for selecting in a referring expression, given the knowledge of the user. As the user acquires new knowledge in the course of a dialogue, the system learned to adapt its behaviour by changing its internal user-model [Jana 14] (Pages 883-920).

2.1.2.3 Data-driven approach

Data-driven models experience recently more and more attention in the Text Generation community. They provide the flexibility and potential to overcome the templates based approaches. Even in the past six years, there have been plenty of studies which show the successes of data-driven models. For example on Dinu and Baroni (2014) recommend that the Text Generation can be performed by using different distributional semantic models [Dinu 14] (Pages 624–633). Another example is from Swanson in 2014 as well, where he performed Text Generation with language modeling on a specified vocabulary constraint [Swan 14] (Page 124). All of this methods and data-driven approaches in general require a lot of training data. The first step in general for such system is to build up an so-called *corpus*. A corpus can be viewed as a kind of template as well, but the way it is used is completely different. The corpus is created through the training of a huge training data set. It consists of a basic set of utterances and it can be further manually extended and redefined [Mani 16] (Page 4). The data-driven model can even be used for a general purpose Machine Translation system, which can respectively even be adapted to a specific domain itself (shown by Wang et al. [Wang 09] Pages 471–477). The corpus contains now a set of vocabulary which can be extended through adding new training instances (e.g. user inputs or synonyms) into the *lexicon*. Extending and diversifying the corpus enhances the quality of the interaction between the system and the user and further enriches the conversational level, mostly the systems response. This can be regarded as a higher abstraction level, used for the modern text summarization systems.

It can be seen in Figure 2.2, that the there is a total amount of 34283 words in the English part of the TownInfo corpus and that there are only 462 distinct tokens. This includes even varied units as proper names and numerals. This example shows an approach to translate from English into French and vice versa. All of the *Advanced Approaches for Text Summarization* (Section 2.4) are fully data-driven. For this reason I provide no more examples.

Category	Num. of units	
	English	French
Size in words	34283	136837
Size in sentences	3151	9000
Number of tokens	462	664
- nouns	205	307
- verbs	81	98
- adjectives	68	76
-adverbs	39	29
Number of patterns	320	284

Figure 2.2: Example Corpus from [Mani 16] Page 103

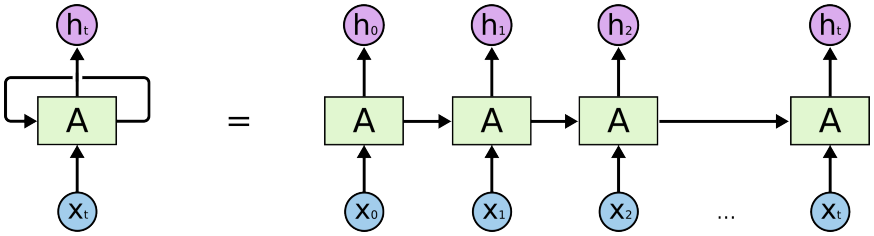

Figure 2.3: Recurrent Neural Network with integrated loops [Olah 15]

The prototype from Chapter 3 is based on a data-driven approach as well, because my text summarizer makes use of Long Short Term Memory cells. More specifically, it is an enhanced modification, namely an Attention model. This data-driven approach will be explained in the next Section *Advanced Approaches for Text Generation*.

2.2 Advanced Approaches for Text Generation

2.2.1 Recurrent Neural Networks

Even though I introduced the neuron in a neural network as a kind of brain cell imitation, the neuron of a basic neural network will forget everything when it is shut down, unlike the brain. Making information persistent is a crucial step towards better performing models. Recurrent neural networks, or RNN, address this issue. They are networks with integrated loops, which allow the information to persist [Olah 15]. The network architecture of the RNN is important, because it denotes the first step into neural text generation and neural text summarization.

Figure 2.3 shows an unrolled Recurrent Neural Network. The input x_t on time step t, is passed to the neural network A. The network looks at the input on this time step and outputs

the hidden state h_t at the same time step t. This loop allows the network to pass information from one time step to another. The picture 2.3 shows, that the learned parameter from input [x] on time step [t] will be passed as additional information to the next time step [t + 1] and so on. For example if a RNN wants to predict the next word in the sentence "Since I am living in Hong Kong .. by now I speak fluent *Cantonese*". The network needs to remember that the target country is Hong Kong to predict the language Cantonese. At each time step t, the hidden state h_t of the Recurrent Neural Network is updated by:

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

where f is a non-linear activation function and x is the input in form of a word. The function f can be in the simplest case a sigmoid function which has either 0 or 1 as output, or the more complex and effective Long Short Term Memory cell, explained in the next Section 2.2.2 [Hoch 97]. The Recurrent Neural Network is trained to predict for example the next word in a sentence or sequence. This prediction is possible due of the learned probability distribution over a sequence. The output at each time step t is a conditional distribution $p(x_t|x_{t-1}, ..., x_1)$.

Theoretically with this approach it is possible to retain information from many time steps ago, but unfortunately, as the time span back grows, RNN's become unable to learn the information from too long ago cells. This phenomenon was explained by Sepp Hochreiter in 1991 [Hoch 91] under the name *vanishing gradient problem*. The solution to this problem is the Long Short Term Memory, short LSTM.

2.2.2 Long Short Term Memory

Long Short Term Memory cells were first proposed by Sepp Hochreiter and Jürgen Schmidhuber in 1997 [Hoch 97]. The LSTM is a special kind of Recurrent Neural Network, because it is able to remember long-term dependencies and informatiom. The goal of the cell is to solve the vanishing gradient problem of the Recurrent Neural Network. Inputs into this cell can be stored for a long period of time, without forgetting them, as in Recurrent Neural Networks. The LSTM is designed to avoid the loss of information (vanishing gradient problem), by intentionally ledging on to certain information over plenty of time steps.

LSTM's can be enrolled the same way like RNN's, but there is a core difference between the Recurrent Neural Network in Figure 2.4 and the Long Short Term Memory in Figure 2.6. The LSTM has four gates instead of one like the RNN. The four gates are:

- Forget Gate
- Input Gate

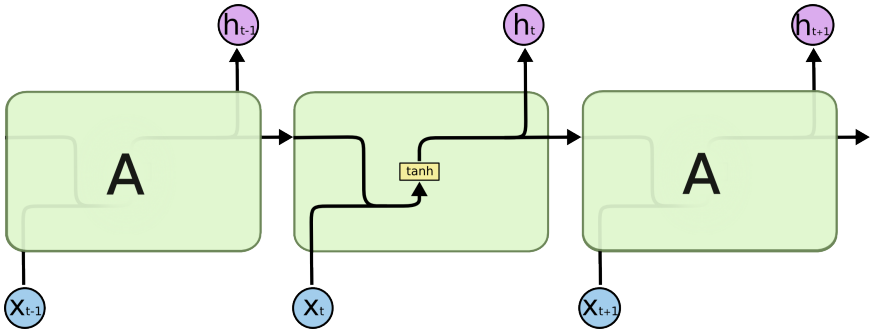

Figure 2.4: The repeating module in an Recurrent Neural Network contains one single layer [Olah 15]

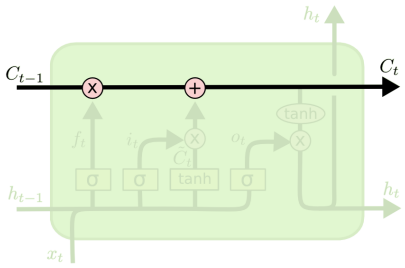

Figure 2.5: Cell State of the Long Short Term Memory which acts as data highway [Olah 15]

- Cell State
- Output Gate

The **Forget Gate** decides what information should be thrown away or kept. Information from the previous hidden state and information from the current input is passed through a sigmoid function. A sigmoid function takes an input and returns high values closer to 1 and smaller values closer to 0. The closer to 0 means to forget the state, and the closer to 1 means to keep the state.

The **Input Gate** updates the cell state. That decides which values will be updated by computing the values to be between 0 and 1 like the Forget Gate. Important information is closer to 1 and 0 means less important.

The **Cell State** is the core of the LSTM. It is the horizontal line shown in Figure 2.5. The cell state acts like the information highway in the cell. With only some minor linear computation, it runs through the entire cell. This way information can pass very easily through the cell.

The **Output Gate** decides what the hidden state of the next LSTM cell should be. The hidden state contains information on previous inputs and it is also used for predictions. The

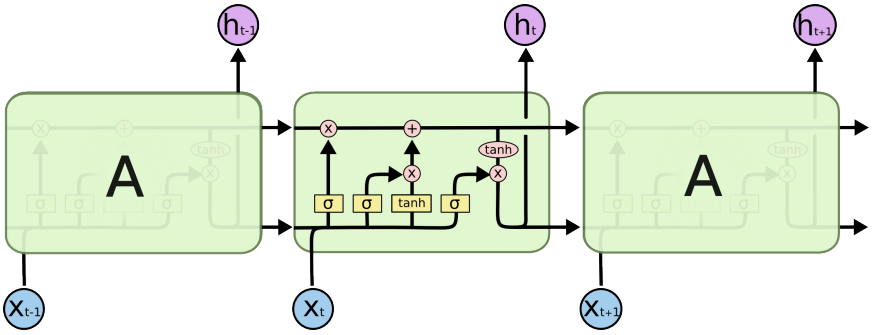

Figure 2.6: The repeating module in an LSTM contains four interacting layers [Olah 15]

hidden state denotes the state which is passed from the output gate on time step t to the input gate for the LSTM cell on time step $t+1$.

The main idea of the LSTM is, that it can decide which information to remove, to forget, which to store and when to use it. It can also decide when to move the previous state information to the next, like the RNN shown in Figure 2.3. Even though many variations of the LSTM occupy the state of the art performance, the LSTM is used in many real business cases in production, like the Google translater or weather forecasting. The Long Short Term Memory paved the way for the sequence to sequence models.

2.2.3 Sequence to Sequence

In the year 2014, Google invented a new way to translate language by learning a statistical model with a neural machine translation approach [Suts 14]. Google called it Sequence to Sequence model [Suts 14], often shortened down to seq2seq, which consists of an encoder and a decoder.

Before that, language translation was originally processed by rule-based systems [Chen 96]. The systems computed their work by breaking down sentences into plenty of chunks and translating them phrase-by-phrase, but this approach created not easily understandable language.

After rule-based systems, statistical models have taken over the. Given a source text in e.g. German (f), what is the most suitable translation into e.g. English (e)? The statistical model $\mathrm{p}(g|e)$ is trained on multiple texts (corpus) and finally outputs $\mathrm{p}(e)$, which is calculated only on the target corpus in English.

$$\hat{e} = argmax_e(e|g) = argmax_e p(g|e)p(e)$$

The formula means, among all Baysian probabilities p($g|e$)p(e), select the pair of words (translation), select the most likely to be the best translation (argmax). Even though this approach produces good results, it looses the wider semantical view, and so it is especially not effective for a good summarization technique.

For the first time, neural networks in form of feed-forward fully-connected neural networks produced such good results, that they replaced all non-network techniques. Affine matrix transformations are stacked together and are followed by non-linearities to the input and each following hidden layer [Beng 03] Page 1141-1142. However, these models require a fixed content length for their calculations, which makes them again not flexible enough to produce human-like translations.

Even if a LSTM (Section 2.2.2) was used to map sequences of words from one language to another, it will most likely produce errors or bad results. A single LSTM cell needs the same input length and output length, which is unrealistic for translating multilingual. For example the English "He is running" translated into German is "Er rennt". The LSTM itself can not translate that, because of the different word length. The Long Short Term Memory cell from Section **??** was invented independently from the sequence to sequence models, but finally three employees of Google published a paper about their approach to make use of the LSTM to create a sequence to sequence model, also called encoder-decoder model. The basic idea is that the encoder converts an input text to a latent vector of length N and the decoder generates an output vector of length V by using the latent encoded vector. It is called a latent vector, because it is not accessible during the training time (manipulating it), for example in a normal Feed Forward Neural Network, the output of a hidden layer in the network can not be manipulated. The initial use of encoder-decoder models was for machine translation.

Technologies for a specific field in the machine learning environment and especially text generation can often be used cross functional. The encoder-decoder model found its way into text summarization and automated email reply by Google [Scie 15] as well. Figure 2.7 illustrates the model for Google's automated email reply.

Figure 2.7 makes use of an Long Short Term Memory cell, which captures situations, writing styles and tones. The network generalizes more flexible and accurate than a rule-based model ever could [Scie 15].

2.2.4 Encoder and Decoder

In the following, the encoder and then the decoder will be explained to have a better insight in how this technology works. The prototype from Chapter 3 is based on this kind of model. As already mentioned a sequence to sequence model is often referred to as a encoder-decoder

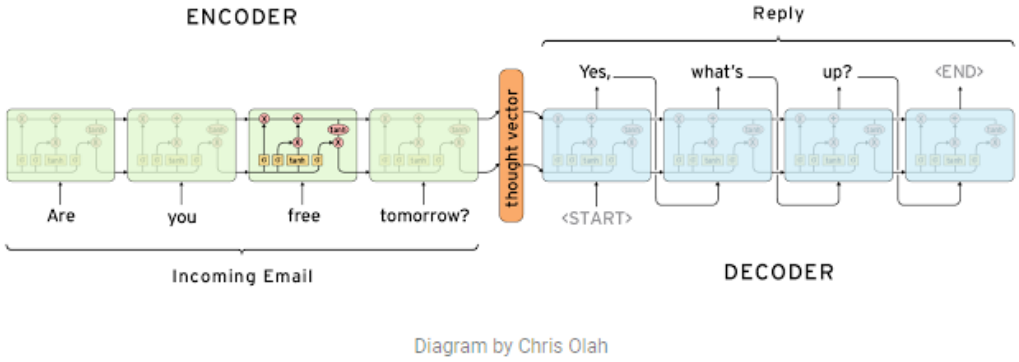

Figure 2.7: LSTM encoder-decoder model for automated E-Mail reply

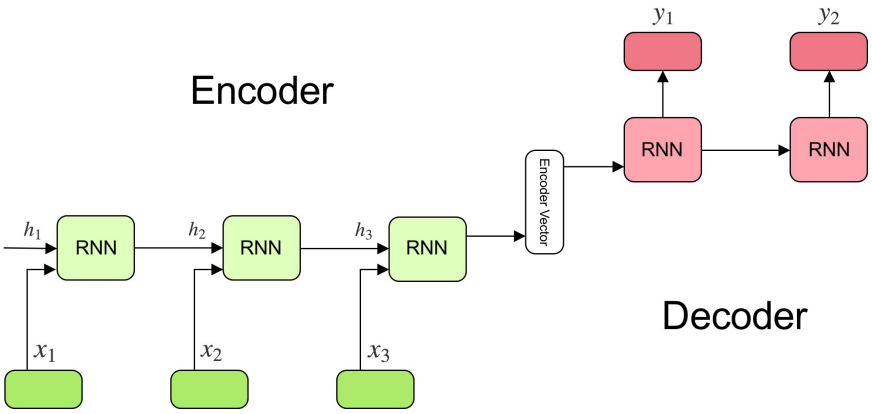

Figure 2.8: Encoder-decoder sequence to sequence model [Kost 19]

model. The sequence to sequence model itself is built using a Recurrent Neural Network or a Long Short Term Memory as explained in the last Section 2.2.3.

Figure 2.8 shows, that the encoder decoder model is built up from actually three parts:

- Encoder
- Intermediate (encoder) Vector
- Decoder

The **Encoder** iteratively integrates the words in a sentence into the hidden state h into the Long Short Term Memory cell. Figure 2.5 shows a single LSTM cell with the input cell state C at the time step *t-1* and the input of the hidden state h at the same time step *t-1*. This is necessary for the cell to compute both the input words, but also the knowledge from prior words. Words are represented as latent vectors in the sequence to sequence models and are stored in a vocabulary table. Each fixed length vector stands for a word in the vocabulary,

Vocabulary Table

aardvark	1.32	1.56	0.31	–1.21	0.31
ate	0.36	–0.26	0.31	–1.99	0.11
...	–0.69	0.33	0.77	0.22	–1.29
zoology	0.41	0.21	–0.32	0.31	0.22

(each row is actually 300 dimensions)

Figure 2.9: Snippet of an example vocabulary table [Muga 18]

for example the vector length is fixed to a dimension of 300. In a simple case, the number of words in the vocabulary is fixed to e.g. 50.000 words, hence the dimension of the vocabulary table in Figure 2.9 is [50000 x 300].

A connection of multiple recurrent units (three in Figure 2.8) where each accept a single element as an input, gains information and propagates it forward to the next cell and accordingly the next time step. In the example of Figure 2.8, the hidden state of *h3* is calculated based on the prior two cells.

The **Encoder Vector** is the last hidden state of all the encoder cells, in this example the encoder vector is located at the output of cell three. The vector tries to combine all of the information from the prior encoded words with the purpose to help the decoder make accurate predictions. Basically, the encoder vector is the initial input for the decoder part of the model.

The **Decoder** unrolls the encoder vector from meaning space into a target sentence. The meaning space (shown in Figure **??**) is a mapping of concepts and ideas that we may want to express to points in a continuous, high-dimensional grid [Muga 18]. The minimum requirement for the meaning space is to consist at least of the last state of the encoder Recurrent Neural Network (encoder vector). The decoder computes a probability distribution for each word in the encoder vector to generate the next state. In the example case, the output is generated by multiplying the hidden state in the encoder vector h by the output matrix of size [300 x 50000]. The product of this matrix multiplication is a vector of size [50,000], that can be normalized with a *softmax* into a probability distribution over words in the vocabulary. The network can then choose the word with the highest probability, because the softmax squeeses all outputs into a summed up probability of 1. For example:

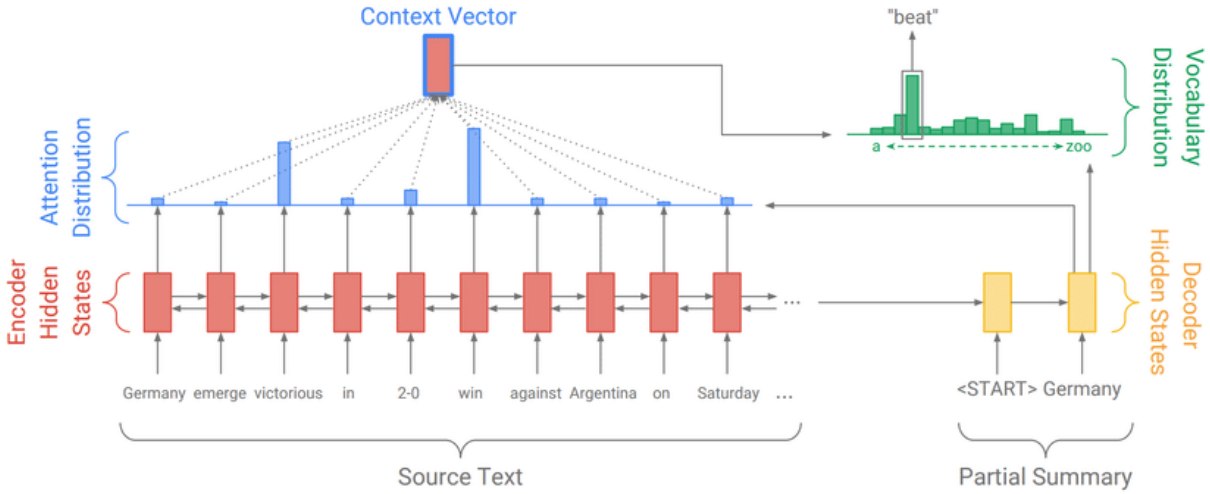

Figure 2.10: Baseline sequence-to-sequence model with attention [See 17]

"Since I am living in Hong Kong, by now I speak fluent ... "

- Cat: 0.01
- running: 0.005
- Cantonese: **0.5**
- Mandarin: 0.3
- French: 0.015

The chosen word is **Cantonese**, because it is the highest probability among all probabilities which are summed up to 100%

2.2.5 Attention

In general, the explanation of the sequence to sequence models just covered the very basic idea of the model. To achieve the state-of-the-art result, not only a single vector can be used for encoding the entire input sequence, but multiple vectors each capable of capturing other information.

In the encoder and decoder model, the length of the state vector h does not change for the input and output. As shown in the example of Section 2.2.3, sentences translated into another language can have a different word length. For the model to automatically adjust the length of the output, is to use the technology called *attention* [Bahd 14] [Vasw 17].

Figure 2.10 shows the basic concept of attention for a summarization task of a news article. The attention model tries to identify relevant words and meanings in the source text to generate novel words. This means to generate words which are not included in the input document, for example using the word *to beat* in the abstractive summary instead of won [See 17].

Attention enables the network to look at all prior encoded states of the words, takes the weighted average probability of the vectors and also uses this as additional information. Sequence to sequence models can be entirely built up from the attention model [Vasw 17].

2.3 Text Summarization Concepts

In the modern era of big data, retrieving useful information from a large number of textual documents is a challenging task due to the unprecedented growth in the availability of online blogs, forums, news, and scientific reports that are tremendous. Automatic text summarization provides an effective and convenient solution for reducing the amount of time it takes to read all the information. The goal of text summarization is to compress long documents into shorter summaries while maintaining the most important information and semantic of the documents [Rade 02] [Mehd 17]. Having the short summaries, the text content can be retrieved, processed and digested effectively and efficiently. Generally speaking, there are two basic approaches for performing a text summarization: Extractive and Abstractive [Mani 99a].

As mention from the Section 2.1, there is a text-to-text and data-to-text approach. The text-to-text method is mostly used in the context where a single document is the only input for generating the text. It can be seen as an extractive approach. On the other side the abstractive approach is actually a data-to-text methods, because it takes into account multiple inputs, such as the text, opinion or another vocabulary.

Jones et al. in 1999 defined and classified text summarization by the following three summarization factors [Jone 98] (Pages 1-12):

- **Input**: single and multi document
- **Purpose**: informative and indicative
- **Output**: extractive and abstractive

This three factors will be explained in the following.

Even though I cannot explain the single state of the art steps at each time in Figure 2.11, I think it provides an good overview on the development of automatic text summarization. Everything started with Peter Luhn from Germany in 1958.

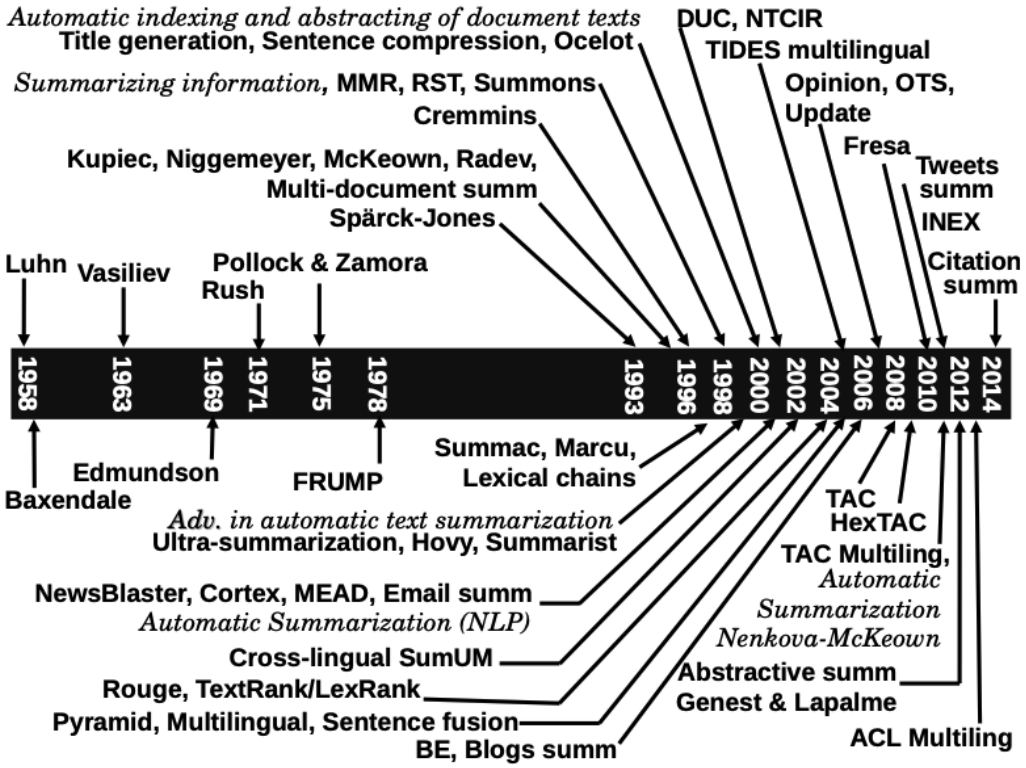

Figure 2.11: Highlights of automatic text summarization [Torr 14] (Page 17)

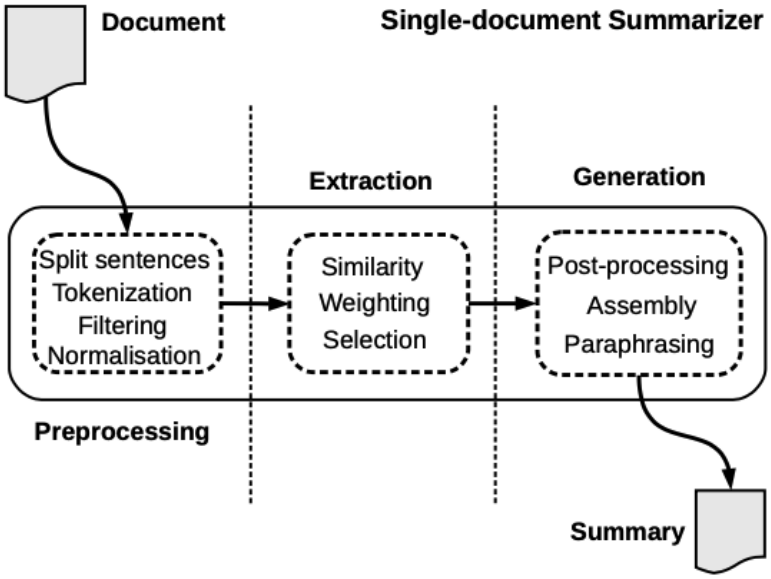

Figure 2.12: General architecture of a extraction-based single-document summarization system [Torr 14] (Page 70)

2.3.1 Input

The input has two crucial variables, that can change the way how to process the summary completely. The input variable denotes whether the input comes from a **single-document** or from a **multi-document**.

2.3.1.1 Single-Document

The most simple task for an automized summarization system is a generic single document summary. This was first introduced by Peter Luhn's work in 1958, an extractive method to summarize a text [Luhn 58]. Even though this first approach was first proposed over 60 years ago, it is still not completely solved. The pipeline for a single document summary is as follows [Torr 14] (Page 69):

- *Preprocessing*: Split sentences into words, cut stopwords (and, that, there, ..) out and filter out punctuation
- *Extraction*: Calculate and combine similarity measure between words and/or sentences, sort and select the sentences
- *Generarion*: Assemble, postprocess and reformulate extracted sentences

Figure 2.12 shows a standardized pipeline, also called *Natural Language Processing Pipeline*. For measuring the similarity between words and sentences, several methods are possible to achieve this goal. It would exceed this thesis if I explain all different methods, but I still want to mention them, because it is the core part of the original pipeline middle-step:

- Latent Semantic Analysis (LSA)
- Graph-based approaches
- Statistical metrics

The **Latent Semantic Analysis** [Deer 90] is a model which allows semantics to be represented from the following ideas: two words are semantically close if they appear in similar contexts and two contexts are similar if they contain semantically close words. The words in a (large) corpus are represented in the occurrence matrix S. The matrix S stores, for every single word in the corpus, the contexts in which the words appeared and additionally also their appearance frequency [Torr 14] (Page 73). This technique measure therefore relationships between different words. After finishing this process, the Latent Semantic Analysis assumes accordingly that words close in meaning occur in similar pieces of text.

Graph-based approaches conduct to represent content of textual information from single documents. There are countless variations of graph-based approaches and I have already used one so far. In Section 2.1.1.5 for the Box 2.1.1.5, the part-of-the-speech tagging is a method in the graph-based approaches. In general, the vertices or nodes are assimilated to semantic collections of words and sentences and the edges of the nodes represent the relations between each words and collections. Another widely used approach is *bag-of-word*. I used that as an example in Section 2.1.2.3 for Figure 2.2. Different unique words are packed together into a corpus and each occurrence of the word is counted and summed together. The ANK algorithm from Lawrence Page in 1998 [Brin 98] (Pages 107-118) paved the way for the success in web page retrieval (scraping): web pages are ranked by their popularity in the network (how often each page is clicked by users), rather than by the amount or quality of their content. This type of algorithm computes the importance of the vertex of the graph, based on the general information gathered from a recursive analysis of the complete graph, rather than a local analysis of a vertex [Torr 14] (Page 77).

2.3.1.2 Multi-Document

Multi-document summarization faces different problems than the single-document summarization. The sentence extraction methods (Latent Semantic Analysis and graph-based approaches) can also be applied to multi-document summarization. The problem of redundancy in the documents is can be always present in multi-document summarization. The typically does not happen in the single-document case. Redundancy has a huge impact on

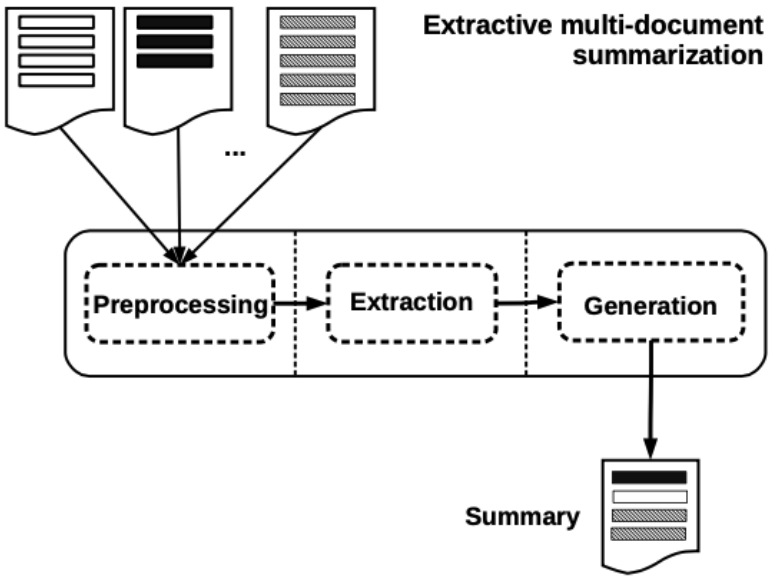

Figure 2.13: Extraction based multidocument summarization system [Torr 14] (Page 110)

the coherence and the cohesion of this new type of extract [Torr 14] (Page 109). Multi-document summarization is basically the extension of single document summarization, but like already said, redundancy is not the only issue. The basic pipeline is shown in Figure 2.13.

Multi-document input are likely to have the same or a similar topic, but it is not necessary. The first automation system was developed by McKeown and Radev in 1995 until 1998 [Rade 98] (Pages 469-500). My case study example Google News is a typical example for multi-document summarization, because Google multiple sources of information (scrape news from different other websites), collect the information and process them through the pipeline (Figure 2.13). The key task is therefore not only detecting and eliminating redundancy, but also notice novelty and ensure that the generated summary is coherent and without missing points [Das 07] (Page 11).

The input for the summarization system are multiple documents such as $D_1, D_2, ..., D_n$, where D is single document and n is the total number of single-document, combined into a multi-document input. The preprocessing step in the multi-document summarization is quite similar, but the same as in the single-document summarization. The four steps for the multi-document are:

- **Sentence segmentation**: Each document D is segmented for itself as $D = S_1, S_2, ..., S_m$, where every S denotes a single sentence in document D. The number of sentences is denoted as m.

- **Tokenization**: Words of each sentence S are tokenized into $T = t_1, t_2, ..., t_k$ k terms t, where every t represents a distinct term occurring in D.
- **Stop word removal**: The most commonly used words in every language are stored in a so-called stop-word-table. Words from that table occurring in a document D are removed. Example words are 'a', 'an' or 'the'. As already mentioned in the single-document summarization
- **Stemming**: converts words back into the base form. For example (houses -> house, running -> run). This is done to avoid redundancy.

After preprocessing the documents into word form, weights are computed to get a sentence informative score. This score is calculated for every sentence accordingly and is used as the input for a chosen optimization algorithm. This happens in the *Extraction* box of Figure 2.13. Like for the single-document, there are multiple methods to calculate extraction weights:

- Abstraction and Information Fusion
- Topic-driven summarization
- Graph Spreading Activation

Abstraction and Information Fusion contains basically of two steps. At first, like in the most methods, a similarity measurement on word or sentence level is computed. When using an abstractive approach (Section 2.3.3), the TFIDF score is commonly used. TFIDF is an information retrieval technique that weighs a term's frequency (TF) and its inverse document frequency (IDF). Every term has its respective TF and IDF score. The product of the TF and IDF scores of a term is called the TFIDF weight of that term [Ramo]. For each term, a vector is calculated that represents matches on the different features. Decision rules that were trained and learned from the data are used to classify each pair of text as either similar or dissimilar. This further feeds a subsequent algorithm that imputes the most related terms in the same topic-theme [Das 07] (Page 13). Once this scores and vectors are computed, the second step *information fusion* starts. This step decides which information should be used for generating the final summary. Rather than just selecting a sentence that holds as a group representative, an algorithm which compares predicated argument structures of the terms. Within each topic, the algorithm needs to determine which terms are used and repeated often enough to be included into the summary.

Topic-driven summarization techniques aim to detect words that describe the topic of the multiple input documents. An advance of the initial idea of Luhn (proposed in Section 2.3.1.1) was to use the log-likelihood ratio test to identify special words known as the *topic signatures* [Luhn 58]. The log-likelihood is often used in statistical approaches for the text summarization. There are two ways to calculate the importance of a sentence. First, as a

function which contains the number of its topic signatures, or as the proportion of all the topic signatures in each sentence respectively. While the first method gives usually higher scores to longer sentences, the second approach measures the occurrences of the topic words [Dunn 93].

Graph Spreading Activation is a similar approach than the graph-based approach from the single-document summarization. The PAGERANK algorithm can be used for this approach, or other alternations like LexRank and TextRank.

2.3.2 Purpose

Types of Summaries

The *informative summary* contains the informative part of the original text. The main ideas from the text should be transmitted, for example the abstract of research articles, where authors try to present the essential core of the reasearch, is an informative summary. Whereas an indicative summary tries to transmit the relevant contents of the original document in such a way, so that the readers can chose documents that match with their interests to read further.

An *indicative summary* is not meant to be a substitute for the original document. The opposite is the informative summary, which can replace the original documents as far as the important contents is concerned and by how much it was shortened down.

The *keyword summary* tries to summarize the text into only keywords. Words will be weighted by their importance and the most crucial ones are selected without caring for the grammar.

The *headline summary* is the type of summary from the case study in Section 1.3. The entire text gets compressed into a single sentence. It is a single line summary.

Generic vs. user-oriented

Generic systems generate summaries that consider all of the given information from the documents. On the other hand try user-oriented systems to produce personalized summaries that concentrate on specific information from the original documents. Like News could only summarize the conservative or right-wing news, if someone only searches for right-wing content.

General purpose vs. domain-specific

General-purpose summarizers can be used across any domains with barely no modification needed. In contrary, domain-specific systems are programmed to process documents for a specific domain, like the research, news or book summarizing domain.

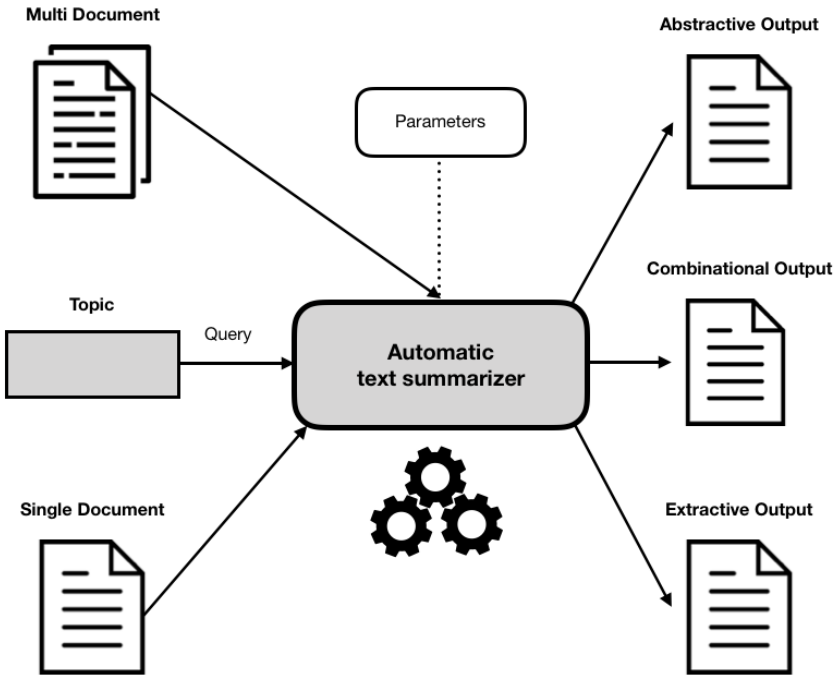

Figure 2.14: Simplified Abstraction Extraction process

2.3.3 Output

During this thesis I have already mentioned the two core differences between text summarization. The extractive and abstractive approach. I use the definition of See et al. from 2017, where he regards the extractive summarizer as an explicit selection method for text snippets inside the single- or multi-document. While the abstractive summarizer generates novel text snippets to describe summaries from a higher point of view by using vocabulary not included in the source input [See 17] (Pages 1073-1083).

Figure 2.14 shows the general model pipeline for either the extractive or abstractive approach. There is also a relatively new combinational approach, which will be explained in Section 2.4 (*Advanced Approaches for Text Summarization*). The topic input into the automatic text summarizer has already been explained in the multi-document Section 2.3.1.2. The parameter input contains for example the compression rate τ. This could a a value like 15%, which means that the length of the original input document will be reduced by 85%.

2.3.3.1 Extractive

The aim of the extractive approach is to give an overview about the source document. This is done by selection fragments of the text (words, sentences or paragraphs) that contain the

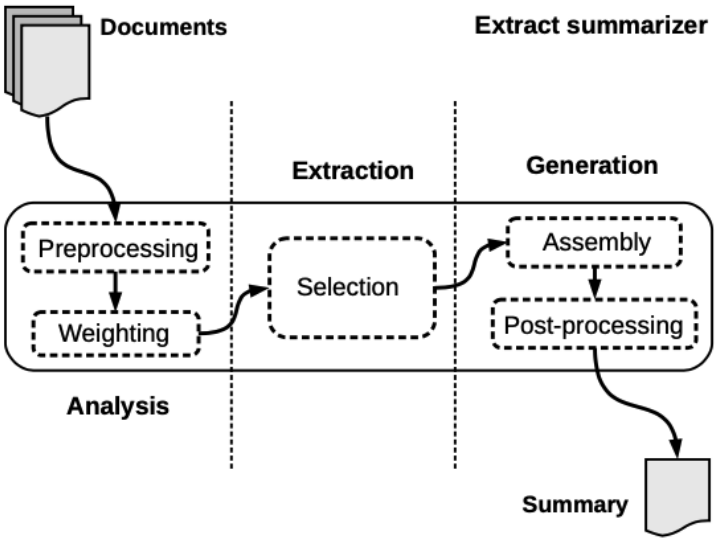

Figure 2.15: General architecture of an extraction-based summarization system [Torr 14] (Page 31)

essential information of the input text or texts. It can be seen as an copy and paste action for the most important fragments. Figure 2.15 shows the basic structure of an extractive summarizer, where the selection part is one of the three categorical types. The extractive approach has still some valid use cases, because:

- **Pros**: The approach is robust, because it uses existing natural-language phrases that are taken directly from the input.
- **Cons**: It lacks in flexibility since it cannot use novel words or connectors. Furthermore it cannot paraphrase like a human could do it. Sometimes the approach even applied wrong grammar.

According to Radev et av. in the 2002, extractive text summarization can be categorized as three different types [Rade 02] (Pages 399-408):

- Surface-level
- Intermediate-level
- Deep parsing techniques

The **surface-level** algorithms scratches on the linguistic easy understandable parts on the text. It cannot detect deep connections and abstractive features. It rather uses certain linguistic elements to detect the most important segments of a document [Torr 14] (Page

32). The mentioned inventor of automatic text summarizer Luhn [Luhn 58] used a surface-level techniques weight the occurrences of words in sentences. This technique is good for headline or keyword summarization.

The **intermediate-level** categorization digs deeper into the meaning of certain paragraphs. It uses linguistic information to find relations between lexical semantic sequences. This approach is more appropriate for extractive text summary than the surface-level approach.

The **deep parsing techniques** approaches make use of deep linguistic techniques that exploit the discursive structure of the input document or documents. It can find good relations in the text and one of the earliest methods was published by Marcu in the year 2002 [Marc 00]. He split the text into discursive units and uses a minimal set of relations, called discourse segmentation. An algorithm weights and orders the elements accordingly. The highest weighted elements will be selected for the summary.

Even though I proposed in the Section 2.3.1 for the single-document and multi-document different methods to compute the weights for the words, the extractive approach and the abstractive approach can be applied for both single- and multi-documents. The most common methods for the extractive approach to calculate weights are:

- Graph-based (both single-document and multi-document) like PAGERANK and Textrank
- Luhn's algorithm [Luhn 58],
- Topic-driven (Section 2.3.1.2)
- Neural Network approach (Section 2.4.1 - *Advanced Approaches for Text Summarization*)

2.3.3.2 Abstractive

The abstractive method for automatic text summarization is one of latest achievements in the research for good automated summaries. When a system is based on an abstractive method, it means that it deeply understands the text and seeks to generate grammatically correct and human-like coherent sentences. One of the first approaches was the *FRUMP* system. It was one of the first systems who used semantic representation for the input text to generate summaries. This system was built in 1982 by Gerald Francis DeJong [DeJo 82] (Pages 149-176).

The core features of FRUMP's algorithm was a complex architecture with the goal to understand documents written in natural language and it had a hard-coded so-called knowledge structure [DeJo 82] to simulate human behavior. But still the module for understanding the

input was conducted by a rather superficial analysis of the text. Frump's algorithm still made a lot mistakes. Taking into account the pros and cons from the extractive approach, the abstractive approach has different strengths and weaknesses:

- **Pros**: Generates summaries in a more fluent way, by using words which are not included in the original input documents
- **Cons**: It is also a much more complex problem for the model to generate coherent phrases and connectors.

Nowadays both he extractive and abstractive methods still are in use. It only depends on the to achieving task what to approach to chose. As I pointed out, the abstractive approach is more complex to use and requires more computitional power. There are even ways to collaborate with the human, as in the **Aided Summarization**:

- Combines automatic methods together with human input.
- The algorithm suggests important information from the input document or documents and the human decide whether to use it or not. It uses information retrieval and text mining.

One of the pioneers for abstractive summarization, Banko et al. (2000) recommends to use a machine translatation model for the abstractive summarization model [Bank 00] (Pages 318-325). I mentioned in the beginning of the thesis that sometimes innovative approaches for one discipline, in the example machine translation, can benefit other disciplines (text summarization). A machine translation model converts a source language into the to translating target language and the text summarization system converts a source document or documents into a target summary.

The most common methods for the abstractive approach are:

- The encoder-decoder model (Section 2.2.4 - *Advanced Approaches for Text Summarization*)
- A neural attention model (Section 2.2.5 - *Advanced Approaches for Text Summarization*)
- A sequence to sequence approach (Section 2.2.3 - *Advanced Approaches for Text Summarization*)

The **encoder-decoder** model from Section 2.2.4 (Encoder-Decoder) was originally invented for the use of machine translation. This model is also commonly used as an abstractive approach for text summarization. The text summarization model which used encoder-decoder model achieved state of the art results on the two sentence-level summarization datasets

DUC-2004 and *Gigaword*. Those two datasets are commonly used for evaluating the results of a summarization on a standardized way. This will be explained in the next section. The encoder-decoder model still faces some problems when using it directly for the text summarization. It needs to modified to work properly:

- Set the focus on the important sentences and keywords.
- Handle the novel, rare or made-up words in source document.
- Handle a really long document.
- Make more human-readable summary.
- Use a large vocabulary.

A **neural attention** model for sentence summarization is another applied abstractive method for text summary. I already explained how an attention model is built up in Section 2.2.5. To overcome issues, the attention model needs to take care of the following points:

- Set the focus on the important sentences abd keywords, like the encoder-decoder, since attention is encoder-decoder based too.
- Handle the novel, rare or made-up words in source document as well.
- Use beam-search to generate summary
- Use the n-gram match term as the loss function

Beam-search is basically just an approximate search strategy to choose the best possible results from all available *candidates* [Budu 17] (Pages 174-178).

N-grams are groups of N consecutive words that can be extracted from a sentence. It can also be applied to characters instead of words. For example [Chol 18] (Page 181):

> *Mario Götze scored 3 times within 51 minutes*
>
> It can be decomposed into the 2-gram:
>
> ('Mario', 'Mario Götze', 'Götze', 'Götze scored', 'scored', 'scored 3', '3', '3 times', 'times', 'times within', 'within', 'within 51', '51', '51 minutes', 'minutes')

Relations of words can be better concluded out of the n-gram method.

The **Sequence-to-Sequence Recurrent Neural Networks** for abstractive Text Summarization. As with the previous two methods, this method has also already been introduced

in Section 2.2.3 and Section 2.2.1. This model uses slightly different approaches and faces therefore partly the same, but more issues:

- Set the focus on the important sentences abd keywords, like the encoder-decoder, since sequence to sequence models are encoder-decoder based too.
- Add enhanced features like named entity tags from Section 2.1.1.5 or TFIDF scores from Section 2.3.1.2.
- Use a large vocabulary
- Use a subset of pre-trained models with a vocabulary [Jean 15] (Pages 1-10).

All of this methods achieving better results than the extractive approaches, but there are still even more advanced approach possible (Section 2.4).

2.3.4 Evaluation

Evaluating automatically generated summaries has always been from the beginning on a really difficult task. New methods needed to be created especially for the summarization discipline. It is divided basically into two different approach to measure the quality of the generated summary [Jone 98] (Pages 1-12):

- The **intrinsic evaluation** directly evaluates the output of a summarized text.
- The **extrinsic evaluation** evaluates summaries based on their performance of the down-stream tasks that the generated summary was computed for.

Intrinsic Evaluation

Until today, there is no single best summarization evaluation method. The manual by hand evaluation is too expensive, as stated out by Lin in 2004 [Lin 04] (Pages 74-81). For that reason he published a method for a cheap automated evaluation metric *ROUGE*. Nowadays this approach is often coupled together with additional human ratings for the best result. ROGUE stands for Recall-Oriented Understudy for Gisting Evaluation. It is a set of methods that can automatically determine the quality of an generated summary by comparing it to a professional human created summary for the same input text. The three most widely ROUGE based methods are [Dong 18] (Pages 2-3):

- **ROUGE-N**: it calculates the percentage of overlapped n-grams with the reference summaries. It requires the one by one matches of all the words in n-grams. It compares the reference and generated summary therefore one by one. The number of words n needs to be predefined

- **ROUGE-L**: it calculates the amount of the most one by one identical words. It therefore automatically identifies the longest in-sequence word overlapping without n being predefined.

- **ROUGE-SU**: is not as straight forward as the other two. It measures the percentage of skip-bigrams (2-grams) and unigrams (1-grams) which overlap. When applying skip-bigrams without constraints on the distance between the words, it usually produces incorrect bigram matches. For that reason the skip distances is limited by a certain number like 4 (ROUGE-SU4) [Lin 04].

An more advanced evaluation metric called **Pyramid** was also published by Nenkova and Passonneau in 2004 [Nenk 04] (Pages 145-152). Based on the assumption that there is no single best summary, but a variety of summaries can represent the input document in the same quality, Pyramid tries to evaluate summaries based on semantically matching content units.

Another well known approach which is originated from machine translation is **BLEU**, which means Bilingual Evaluation Understudy. BLEU can also be applied for evaluating text summaries. BLEU is calculated in general on the n-gram co-occerance between the generated summary and the ideal human written summary. It measure how many of the words or n-grams in the machine generated summary appeared in the reference summary from a human. Not to be mixed up with ROGUE, which counts how many of the words or n-grams in the human reference summary appeared in the machine generated summary. Both approaches are relatively similar and can be used for evaluating, but they are not the same.

Extrinsic Evaluation

This approach has three most commonly methods, namely [Stei 09] (Pages 10-11):

- **Document categorization**: the quality of a summary can be measured by its suitability for surrogating a full document for categorization. This means, is the summary categorizes in the correct way? For example is the summary of the topic international news?

- **Information retrieval**: a summary should capture the core points of a document, then an information retrieval machine indexed on a set of summaries should generate a good summary.

- **Question Answering**: multiple choice with a single answer to be selected should measure how many of the questions randomly chosen people answered correctly under different conditions based on the generated summary.

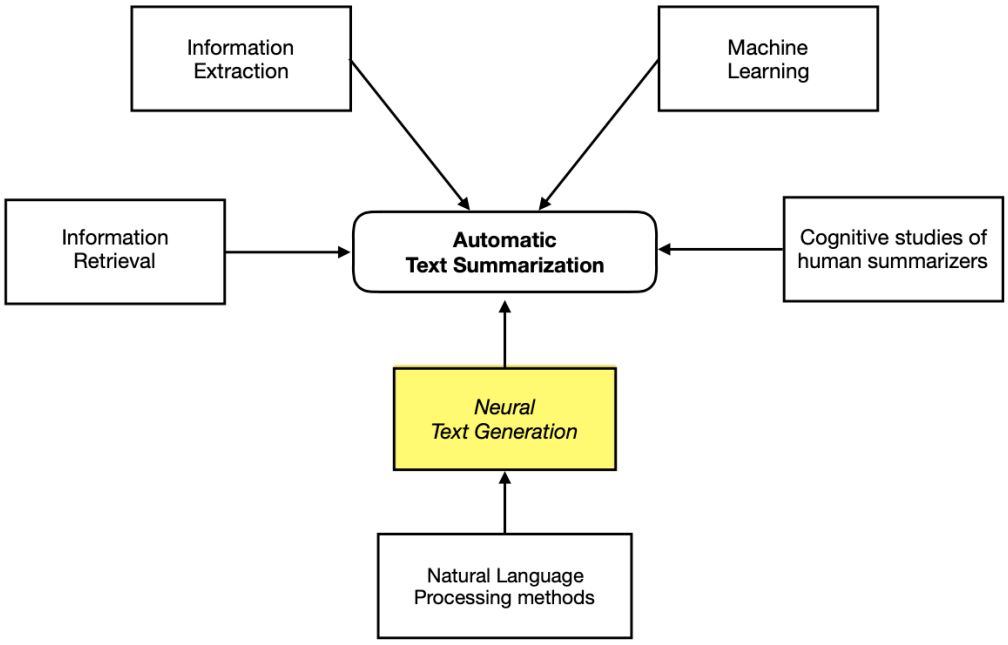

Figure 2.16: Research fields with influence on the development of text summarization

2.4 Advanced Approaches for Text Summarization

Since deep learning received more and more attraction, neural-based summarizers had an considerable influence on automatic summarization. Compared to the traditional models (extractive and party abstractive models), neural-based models achieve better performance with by relying less on the human intervention if the training data is big enough.

The four white boxes in Figure 2.16 have been introduced throughout this thesis. The next step to an even better (human-like) summary is the use of neural text generation, the state of the art approach. In the following, combinational approaches and the reinforcement approach will be introduced. Those methods rely completely on neural networks (neural text generation), but achieve currently in 2020 the best state of the art results. Neural-based approaches are promising for text summarization in terms of the generated human-like summary when large data sets are available for training. Still many challenges and issues with neural-based models remain unsolved. Future research directions such as the combinational approach or even adding the reinforcement learning are still in research.

2.4.1 Combinational Approach

- **Pointer-Generator Network**

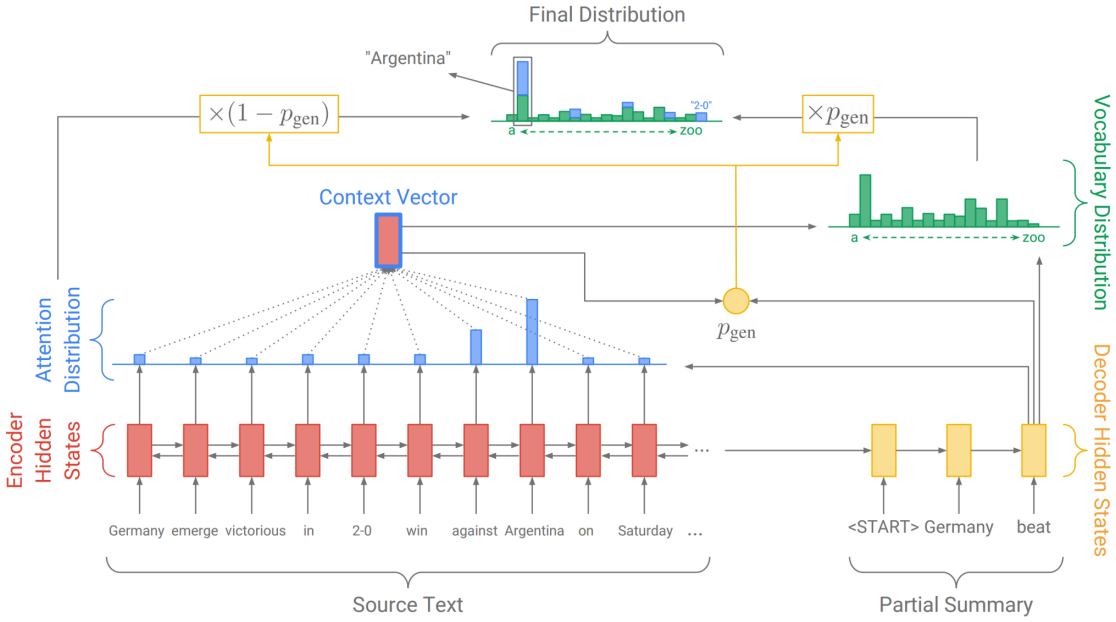

Figure 2.17: The image describes the combination of the weights and the vocabulary distribution [See 17]

- **Extract then Abstract model**

The **Pointer Generator Network** proposed from See, Manning and Liu in 2017 [See 17] makes use of an attention based distribution to generate a probability.

Figure 2.17 is based on the attention model (Figure 2.10 from Section 2.2.5). This model switches the decoder(generator) and the pointer network by a probability p_{gen}. Furthermore it combines the vocabulary distribution and attention with p_{gen} and the (1 - p_{gen}) weight (Figure 2.17). The reason for choosing this probability multiplications exceed this bachelor thesis. The point for me to include it is to introduce the current state of the art models

The **extract then abstract model** uses extractive model first to select the sentence from a document or documents and then secondly it adopts the abstractive model to the selected sentences. Basically saying, the model combines a query focused extractive and abstractive model. It extract the important sentences with an extractive approach and calculate then the relevance score for each word according to the query uses that as input for a pre-trained abstractive model.

2.4.2 Transfer Learning

It is effective to reuse already learned models to make a new text summarization model. That is called *Transfer Learning*. It allows to make a model by a tiny amount of new data and in a short period of time. This feature leads to domain-specific summarization.

BERT is the representative model that enables getting good representation of sentences. Several methods are proposed to create summaries by BERT.

Fine-tune BERT for Extractive Summarization

- Use the pre-trained models
- Get good sentence representation.
- BERT generates token based features. Therefore there is a need to convert token based features to sentence based representation. Liu and Lapata (2019) [Liu 19] use *Segmentation Embedding* to notice sentence boundaries and use the first token of the segmentation as the sentence embedding.

Pretraining-Based Natural Language Generation for Text Summarization

- How to use the pre-trained model?
- Getting good sentence representation and refine a generated sentence.
- BERT is trained to predict the masked token, so this approach can not generate a sequence. Haoyu et al. [Haoy 19] use this method to generate the first summarization by an ordinary transformer model and then drop some tokens to let it fill by the BERT algorithm. The final summarization is created by the so-called input BERT representation and refined sentence representation made by BERT as well.

Chapter 3

Prototype

In this third section of my thesis I finally present my self programmed prototype. The State of the Art chapter was structured in such a way, that it focuses on the necessary information for understanding my prototype. Hence I only gave some additional information about techniques which produce even more accurate results, but this is out of scope for my thesis. This section is divided into four sections, starting with the objective (Section 3.1) of this prototype. I propose my requirements for this prototype and what I expect to achieve from it. The next step is the technical concept (Section 3.2), which models the data flows and processes inside the program. After I explain which steps my algorithm goes through, I present snippets of my code in Section 3.3 to further illustrate it. This code will be evaluated in Section 3.4.

3.1 Objective

"I don't have time to read the entire report, please give me a brief summary". Many years ago the challenge was to find the right information for a specific tasks, nowadays the internet provides more information than anyone could ever read. Depending on the type of document or article, it is not always necessary to read it from the beginning to the end. Throughout my thesis I often used my case study *News Headline Summarization from Google* for illustrating certain concepts. I chose a different topic for my case study than my actual prototype to show another kind of task which automatic text summarization is capable of.

I chose the **Amazon Fine Food Reviews** provided from *Stanford Network Analysis Project* hosted on the website Kaggle [1]. This website is commonly known for providing data-sets from private and organizational publishers.

The project's objective is to build and train a model that can compute relevant summaries for reviews written about fine foods sold on Amazon.

[1] https://www.kaggle.com/snap/amazon-fine-food-reviews/

This dataset contains around 500.000 entries with each entry containing one review from the amazon fine foods and its summary. The data is collected for 8 years from October 2012 until now.

3.2 Technical concept

An example review needs to pass from the given data-set file to the predicted summary a lot of processes and transformations. The technical concept illustrates all necessary steps without yet going into the programming detail itself. The purpose of this section is to take all the previous explanations into a combined example and show the entire process from the beginning to the end.

The processing steps follow in chronological order:

- Data pre-processing
- Build the Model
- Train the Model
- Generating Summary

This steps will be explained in the following Subsection.

3.2.1 Data Pre-processing

The computer does not understand words. It is not possible to feed the words from a review subsequently into a recurrent neural network and expect that the algorithm will learn it somehow. Performing pre-processing is a crucial step before feeding inputs into the neural network model. The data must be normalized into a clean and not messy way. The following is an example from the original dataset:

Review
I can remember buying this candy as a kid and the quality hasn't dropped in all these years. Still a superb product you won't be disappointed with.

Summary
Delicious product!

For all the words occurring in the review and summary a vocabulary dictionary needs to be created. This dictionary can be used to convert the words into numbers and respectively

numbers is what the recurrent neural network wants as an input. For creating the two vocabulary dictionaries it needs to be carefully though of how to handle the data. If the same word e.g. *Still* and still is once with a capital letter and the second time without, the encoding (*UTF-8*) is different and hence the computer regards this as two different words. Furthermore the text can include punctuation, quotation marks or parenthesis. Therefore I defined a chronological order of text transformation steps to normalize the text:

- Convert all words into lowercase
- Remove occurring HTML tags
- Contraction mapping
- Remove (‘s)
- Remove any text inside the parenthesis ( )
- Eliminate punctuations and special characters
- Remove stopwords
- Remove short words

Contraction mapping describes the process of mapping contracted words into their original two words. For example *haven’t* will be mapped back to *have not*. This is important, because only this way the algorithm can detect the word *not* to be negative connotated. If a review says something is not bad, then the algorithm can summarize it by computing the word good. This relation can only be learned by the algorithm by providing it all the necessary words.

Remove stopwords limits the vocabulary size, which reduces the models training time dramatically. I have already explained stopwords in Section 2.3.1.1. Those are words like and, that or there.

Remove short words, because they most likely don’t play a vital role in predicting the output. Words like it, in or at not necessarily appear in the stoowords dictionary. For this reason I remove them manually, because they almost don’t affect the text’s meaning, but after removing the reduce the training time further.

After applying all of the pre-processing steps, the example will look as the following:

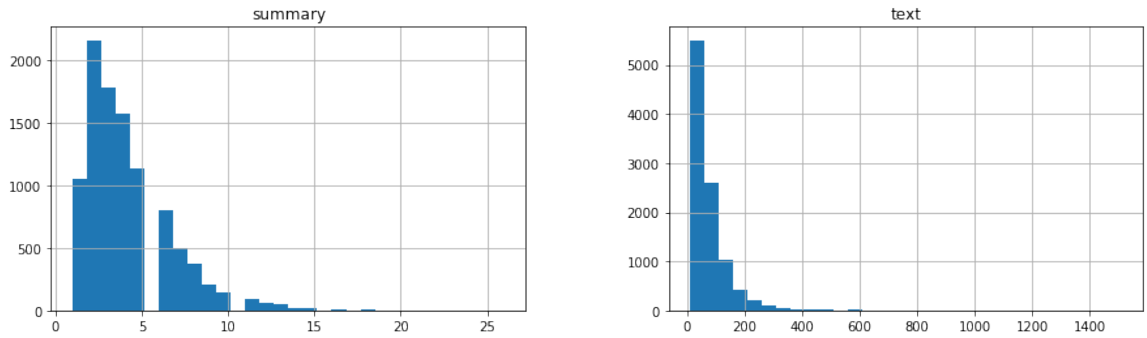

Figure 3.1: Distribution of the sequences to estimate the maximum length for the review and summary

Cleaned Review
remember buying candy kid quality dropped years still superb product disappointed

Cleaned Summary
delicious product

The data is now clean and normalized. Furthermore it can be check whether some data is duplicated and remove this data respectively.

In order to build the mentioned vocabulary, the text needs to be **Tokenized**. This means selecting all the single words of a sentence and store all the words uniquely in the vocabulary for the review and the other vocabulary for the summary. The only thing left is to compare the word counts of all sentences and take the median value for the maximum review length and the maximum summary length as shown in Figure 3.1. This is important, because a fixed length is mandatory for building the model. All of the words are now replaced by their dictionary number representation and if a review is shorter as the maximum length, the missing data points are filled with zeros.

Tokenized and Padded Review
array([[678, 142, 360, 1017, 87, 3724, 108, 55, 1893, 9, 243, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]], dtype=int32)

Tokenized and Padded Summary
array([[1, 21, 19, 2, 0, 0, 0, 0, 0, 0]], dtype=int32)

This is the necessary format for building up a model.

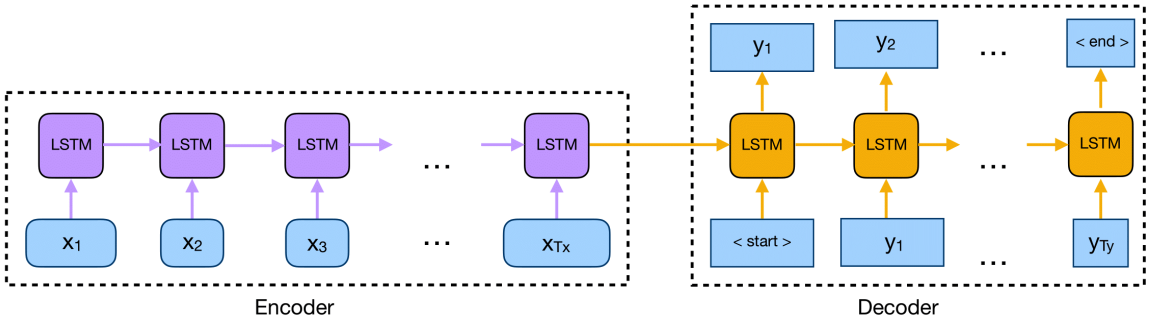

Figure 3.2: Many to many Sequence to Sequence LSTM model

3.2.2 Building the Model

The model follows an abstractive approach (Section 2.3.3.2) and is based on the Advanced Approaches for Text Generation (Section 2.2). Since the objective is to build a text summarizer, the input is the fixed length sequence of words and the output is a shorter, but also fixed length sequence of words (Figure 3.2).

The mathematical notations for Figure 3.2 are shown in the box below.

$[\boldsymbol{x_1}, \boldsymbol{x_2}, \boldsymbol{x_3}, ..., \boldsymbol{x_{Tx}}]$ where each x represents one word of the input sequence,
e.g. $\boldsymbol{x_1}$ = remember

$[\boldsymbol{y_1}, \boldsymbol{y_2}, \boldsymbol{x_3}, ..., \boldsymbol{y_{Ty}}]$ where each x represents one word of the input sequence,
e.g. $\boldsymbol{y_1}$ = delicious

$\boldsymbol{T_x}$ represents the fixed length of the input sequence

$\boldsymbol{T_y}$ represents the fixed length of the output sequence

The reason for choosing the LSTM over a basic recurrent neural network architecture is, that the generated summary can be sometimes quite long (up to 20 words) and the recurrent neural network cannot catch up with dependencies of this sequence length. Figure 3.3 illustrates the overall view of my model. As explained in the last section, if a input or output has less words than the maximum length, the missing words are denoted as zero values. This allows the algorithm to not compute any further prediction, because it is a zero multiplication. The encoder decoder architecture is mainly used when the input and output length vary from each other.

Figure 3.2 shows the general sequence to sequence model, whereas Figure 3.3 illustrates the perspective of the text summarizer.

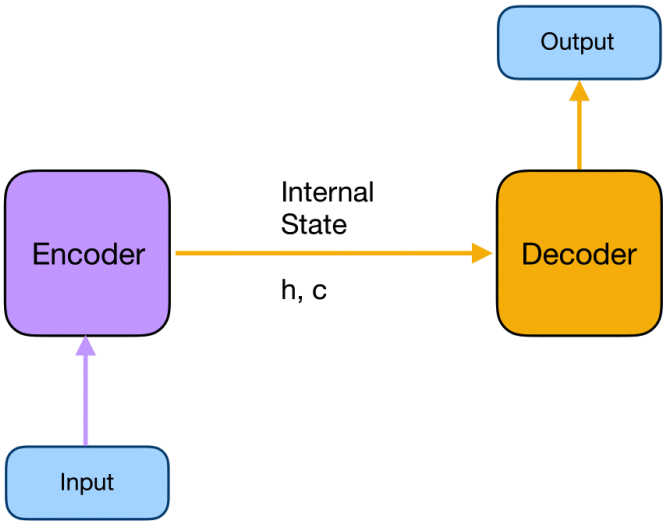

Figure 3.3: Perspective of the text summarization

The mathematical notations for Figure 3.3 are shown in the box below.

h represents the input (update) gate, which is responsible for adding new information into the cell

c represents the cell state, which is the horizontal line between multiple LSTM cells

3.2.3 Training the Model

For the training phase, first of all the encoder and decoder need to be set up individually from each other. The model predicts the next word based on the current word (or the start). To achieve this, the prediction is one time step in the future.

The **Encoder** in Figure 3.4 uses LSTM cells to read all of the sentences from the data-set in. Each sentence will be read in at a time and further each word of the sentence will be read for each at a time step. Therefore the LSTM has a one word input per time step and captures the contextual information of the sentence out of it. The h and c in Figure 3.4 are the same as the ones from the previous Figure 3.3.

The encoder and the decoder are two different LSTM architectures. The initial state of the encoder LSTM is a zero matrix and the finally computed output state of the encoder LSTM is the initial state of the decoder LSTM.

The **Decoder** is based on a LSTM network as well, which has the target sequence as a word-by-word input and predicts the same sequence offset by one time step. The decoder

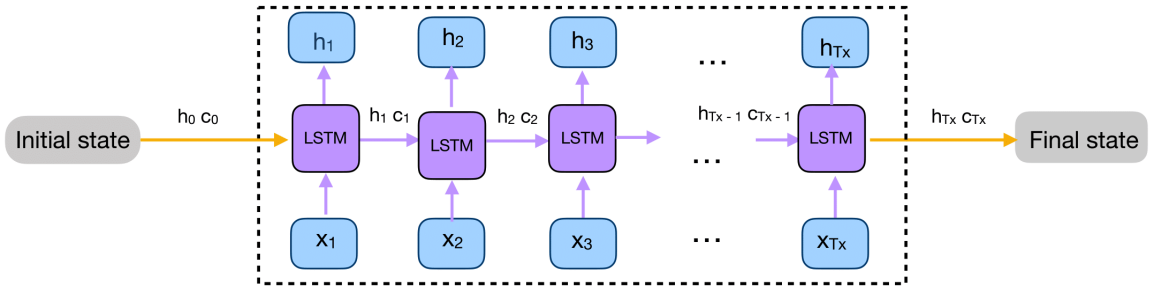

Figure 3.4: Encoder of the LSTM

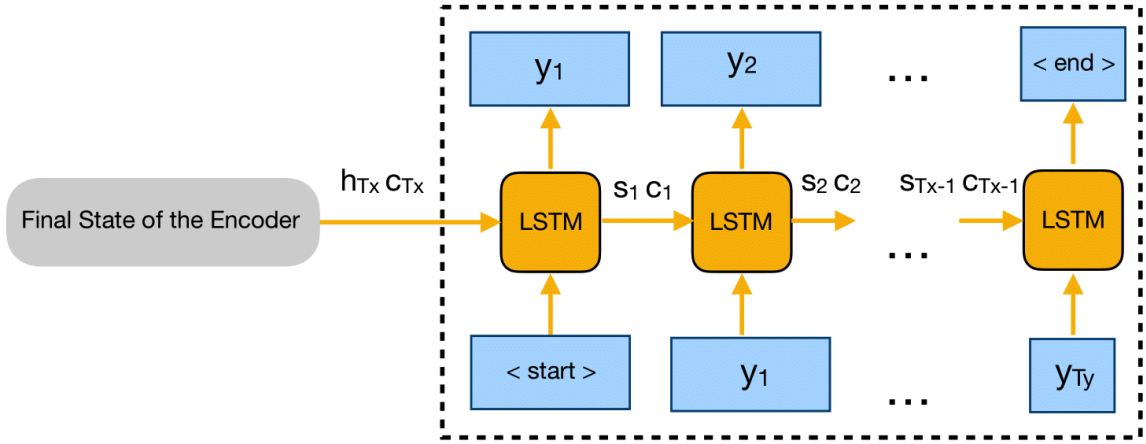

Figure 3.5: Decoder of the LSTM

is trained to predict the upcoming word in the sequence with respect to the previous word. The *start* and *end* boxes in Figure 3.5 represent an appended string to the beginning and ending of the original summary. This is done to create the one time step offset.

The target sequence is unknown during the decoding part of the test sequence. Target sequence is predicted by passing the first word into the decoder which would is always the *start* token. And the *end* token signals the end of the sentence, followed by potentially zero values.

3.2.4 Generate the Summary

This part is also known as the *Inference Phase*. After training on a selected amount of reviews and summaries, the model is tested on new source sequences where the target sequence is unknown. For this to work, a inference architecture to decode a test sequence is necessary. The inference goes through various processes:

- 1) Encode the review and initialize the decoder with internal states of the encoder

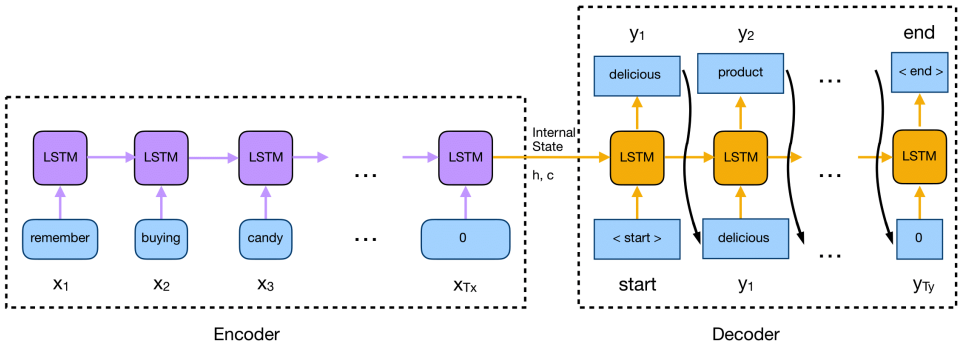

Figure 3.6: Inference architecture to decode a sequence

- 2) Pass the *start* token as an input to the decoder
- 3) Compute the decoder for one time step with the stored internal states
- 4) The output at each time step is the probability for the next word. The maximum probability word will be selected
- 5) Pass the selected word as an input to the next decoder time step and update the internal states with the current time step
- 6) Repeat steps *3 – 5* until the *end* token is generated with the highest probability

Figure 3.6 shows the architecture for the inference phase. The test sentence goes through all the initially trained weights (states h, c) of the model. There is no more training necessary, just the calculations of the test sequence with the internal state vectors to predict the output sequence. After a certain prediction length, the probability for the *end* token to occur is the highest and if there is remaining space for maximum summary length, it will be filled up with zero values again.

Even a LSTM cell has limitations when it comes to sentence length. Since summaries can be quite long (my maximum length is 80 words), there need to be made further improvements for the algorithm in order to work properly. It is difficult for the encoder to memorize long sequences and output a fixed length vector. For this reason I have introduced the attention mechanism in Section 2.2.5. The intuition for attention is:

> *How much attention do we need to pay to every single word in the input sequence for generating a new word at time step* **t***? That is the primal intuition behind the attention mechanism.*

With this mechanism it is possible for the decoder to first look on all the words from the encoder output and decide to which parts of the sentence the decoder wants to focus

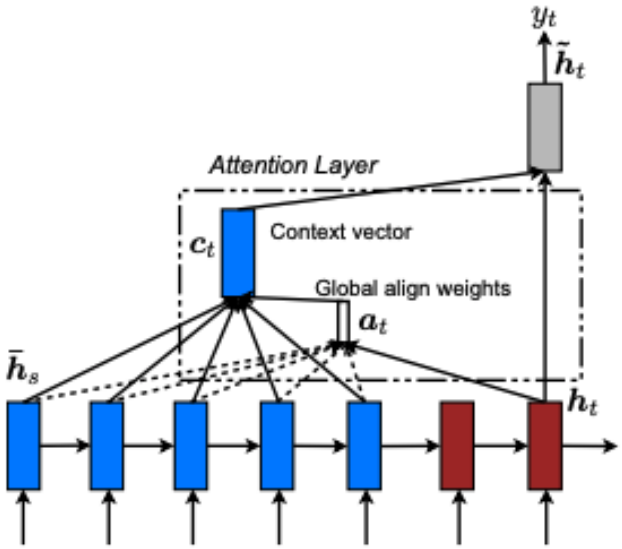

Figure 3.7: Global Attention Model from [Minh 15]

time-by-time. The attention model is placed on top of the LSTM cell and requires additional computation. For my model I am using the **global attention model** from [Minh 15] shown in Figure 3.7, where h denotes the same weights as Figure 3.3 and a is the attention parameter, calculated for every word at every time step t.

h is the hidden state trained on the encoder

t is the time step which represents always a single word

a is calculated $\boldsymbol{T_y}$ times at every time step **t**, so for the attention model we have **a** * $\boldsymbol{T_y}$ * **t** new parameters

3.3 Implementation

I will not provide the entire code in this section, but I will introduce the most crucial aspects of the code. I used python version 3.7.6 and I programmed it entirely in the **Anaconda**[2] IDE (Integrated Development Environment). This IDE makes it especially easy for programming data science related topics like text summarization in python, because it can preview each cell one-by-one. Anaconda is for free and a open source platform. The python packages I used for this project are listed in the following Listing:

[2]https://www.anaconda.com/

```
1  import numpy as np
2  import pandas as pd
3  import re
4  from bs4 import BeautifulSoup
5  from keras.preprocessing.text import Tokenizer
6  from keras.preprocessing.sequence import pad_sequences
7  from nltk.corpus import stopwords
8  from tensorflow.keras.layers import Input, LSTM, Embedding, Dense,
                                       Concatenate, TimeDistributed
9  from tensorflow.keras.models import Model
10 from tensorflow.keras.callbacks import EarlyStopping
```

Numpy takes care of the computations and data structures. This package is implemented in such a efficient and effective way, so that it saves up a lot of computation time when using this package instead of the default python data structures and mathematical operations.

Pandas is responsible for most of the data pre-processing. The amazon reviews are loaded into a so called Pandas *DataFrame*. Within the DataFrame the data goes through all the necessary pre-processing steps as explained in Section 3.2.1. Within the pre-processing step *Remove occurring HTML tags*, the package **BeautifulSoup** is a useful tool for achieving this step. **NLTK** is short for Natural Language Toolkit. Many functions regarding Natural Language Processing can be used from this package, but I used it only for downloading the *stopword* for the second last pre-processing step *Remove stopwords*.

Keras is the main package for building the model in Section 3.2. Keras is built up from the package **Tensorflow**. TensorFlow has a high-level API for building and training deep learning models. It can be used for doing fast prototyping or state of the art research [3]. Since the new update from Tensorflow version 1 to version 2, Keras is normally integrated into the project with Tensorflow (tensorflow.keras indicates Keras is downloaded within Tensorflow). Keras includes most of the required building blocks, such as the LSTM cell or the Embedding Layer which is responsible for mapping the words into vectors. In the following is the encoder part of the trainable model:

[3] https://www.tensorflow.org/guide/keras

```
# Encoder
encoder_inputs = Input(shape=(max_len_text,))
enc_emb = Embedding(x_voc_size, latent_dim,trainable = True)
(encoder_inputs)

#LSTM 1
encoder_lstm1 = LSTM(latent_dim,return_sequences = True, return_state=True)
encoder_output1, state_h1, state_c1 = encoder_lstm1(enc_emb)

#LSTM 2
encoder_lstm2 = LSTM(latent_dim,return_sequences = True, return_state=True)
encoder_output2, state_h2, state_c2 = encoder_lstm2
(encoder_output1)

#LSTM 3
encoder_lstm3  =LSTM(latent_dim, return_state=True,
return_sequences = True)
encoder_outputs, state_h, state_c = encoder_lstm3(encoder_output2)
```

The encoder is initialized with the mentioned maximum length of the summary. The Embedding layer creates the vectors out of the input reviews with the limitation of the maximum unique vocabulary word length. The *latent_dim* parameter denotes the amount of units used for the embedding, in my case I used a value of 500. The encoder is further built up from three LSTM building blocks which is called a *Stacked LSTM*, that has multiple layers of LSTM's stacked on top of each other. This can lead to a better representation of the sequence. The two used parameters for the LSTM are explained in the following:

- **Return Sequences = True**: When the return sequences parameter is *True*, the LSTM produces the hidden state h and cell state c for every time step
- **Return State = True**: When the return state = *True*, the LSTM produces the hidden state h and cell state c only from last time step

Those parameters are used the same way for the decoder part of the model. The decoder takes, as explained, the output of the encoder as input into its model. The decoder uses a single LSTM block and also the Attention building block. The only difference is that the attention block is not officially support by Tensorflow, hence it needs to be important through a third-party library or important through an extensible python file. For my model I am using the Bahdanau attention implementation introduced in the paper *Neural machine translation by jointly learning to align and translate* [Dzmi 16].

The output of the decoder is fed into the *TimeDistributed* layer, which creates the time steps t. The attention layer is concatenated with the decoder layer and the defined model at the end of the following code combines the encoder, attention and the decoder together:

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 80)]	0	
embedding (Embedding)	(None, 80, 500)	40049500	input_1[0][0]
lstm (LSTM)	[(None, 80, 500), (N	2002000	embedding[0][0]
input_2 (InputLayer)	[(None, None)]	0	
lstm_1 (LSTM)	[(None, 80, 500), (N	2002000	lstm[0][0]
embedding_1 (Embedding)	(None, None, 500)	10933500	input_2[0][0]
lstm_2 (LSTM)	[(None, 80, 500), (N	2002000	lstm_1[0][0]
lstm_3 (LSTM)	[(None, None, 500),	2002000	embedding_1[0][0] lstm_2[0][1] lstm_2[0][2]
attention_layer (AttentionLayer	((None, None, 500),	500500	lstm_2[0][0] lstm_3[0][0]
concat_layer (Concatenate)	(None, None, 1000)	0	lstm_3[0][0] attention_layer[0][0]
time_distributed (TimeDistribut	(None, None, 21867)	21888867	concat_layer[0][0]

Total params: 81,380,367

Figure 3.8: Model summarization with parameters

```
# Decoder.
decoder_inputs = Input(shape = (None,))
dec_emb_layer = Embedding(y_voc_size, latent_dim, trainable=True)
dec_emb = dec_emb_layer(decoder_inputs)

#LSTM using encoder_states as initial state
decoder_lstm = LSTM(latent_dim, return_sequences = True, return_state=True)
decoder_outputs,decoder_fwd_state, decoder_back_state = decoder_lstm(dec_emb
                                        , initial_state = [state_h, state_c])

#Attention Layer
attn_layer = AttentionLayer(name='attention_layer')
attn_out, attn_states = attn_layer([encoder_outputs, decoder_outputs])

# Concat attention output and decoder LSTM output
decoder_concat_input = Concatenate(axis=-1, name='concat_layer')([
                                        decoder_outputs, attn_out])

#Dense layer
decoder_dense = TimeDistributed(Dense(y_voc_size, activation='softmax'))
decoder_outputs = decoder_dense(decoder_concat_input)

# Define the model
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
```

The summarized model looks now like Figure 3.8 with its 81.380.367 parameters.

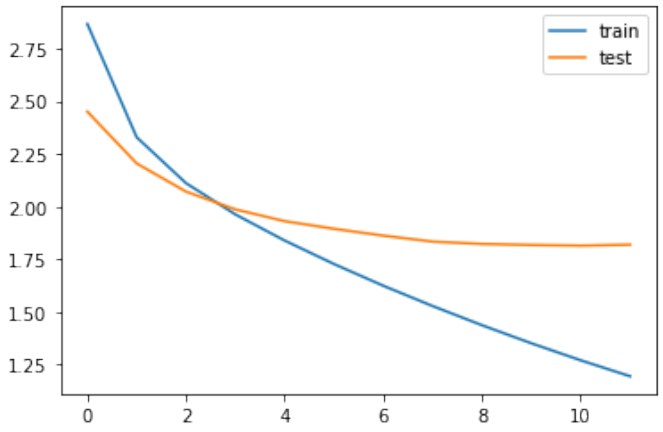

Figure 3.9: Training and validation loss during the training phase

3.4 Evaluation

The model and output can be evaluated on multiple ways as explained in Section 2.3.4. A common approach for evaluating a model is by saving the training and testing loss during each training epoch. The loss function used for this model is the *sparse_categorical_crossentropy*, because it converts the integer sequences to a one-hot encoded vector. This saves up a lot of memory issues. One-hot encoding transform arrays into only ones and zeros. The stored losses of each epoch can be plotted into a graph shown in Figure 3.9.

Loss

Figure 3.9 shows, that the training can be stopped at epoch 10, because the loss is beginning to increase again, which is bad, because the loss needs to be as small as possible.

Keras provides a prediction function for the model which can be used for a review to generate the summary. The output of the prediction is an integer array which can be mapped back into the original, but cleaned words. The following table presents the computed summaries with the input reviews and the original summaries as a check for quality of the summaries:

Predictions

Those predictions have been made with a training of 224814 samples and was validated on 24980 samples for the validation (testing) loss. After epoch 12 the training stopped due of early stopping, because the loss increased again instead of further decreasing. This can be caused by many reasons.

Improving the models performance

Cleaned Review	Cleaned Summary	Generated Summary
ordered chips found salty dry huge amount spices ball one bags opened	too salty and dry	too salty
found tea favorite movie theater found perfect tea guests everyone loves makes love	at the movies and home	love it
dogs special diet treats feed favorites cause problems	must be good	my dogs love these
delicious sherry flavor salad dressing great used marinade give try sweet balsamic tart red wine vinegar	yummy sweet sherry vinegar	love these
received medium roast receive correct coffee shown picture disappointed suppose ill try lot trouble return	wrong coffee received	coffee received

Table 3.1: This table shows some example outputs for the training data from my trained model

- Increasing the training data. Since I used only half of the available training data, additional 250.000 data entries can be used for the training. Nevertheless, this requires a lot of computing power and every epoch can take up even on a strong GPU up to 2-3 hours with 500.000 values.
- There exist enhanced LSTM models like the Bidirectional LSTM, which is capable of capturing the context from the forward and backward process of the sentence before feeding it into the decoder. hence this takes the double computation time as well.
- Using the Beam-Search strategy from Section 2.3.3.2 can lead to better results
- Pointer-Generator Networks from Section 2.4.1 can improve the models performance dramatically as well, but it was out of the scope for me to implement it, because it requires a lot of knowledge to implement it properly.

Chapter 4

Generation of transferable knowledge

Modular expandability of my project. Classification in social context

List of Figures

List of Tables

List of Listings

References

[Anja 10] J. V. Anja Belz, Eric Kow. "Generating referring expressions in context: The GREC task evaluation challenges. In Empirical Methods in NLG". *Springer*, 2010.

[Bahd 14] D. Bahdanau, K. Cho, and Y. Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". 2014. cite arxiv:1409.0473Comment: Accepted at ICLR 2015 as oral presentation.

[Ball 15] M. Ballesteros, B. Bohnet, S. Mille, and L. Wanner. "Data-driven sentence generation with non-isomorphic trees". In: *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 387–397, Association for Computational Linguistics, Denver, Colorado, May–June 2015.

[Bank 00] M. Banko, V. O. Mittal, and M. J. Witbrock. "Headline Generation Based on Statistical Translation". In: *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, pp. 318–325, Association for Computational Linguistics, Hong Kong, Oct. 2000.

[Barz 04] R. Barzilay and L. Lee. "Catching the Drift: Probabilistic Content Models, with Applications to Generation and Summarization". In: *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*, pp. 113–120, Association for Computational Linguistics, Boston, Massachusetts, USA, May 2 - May 7 2004.

[Beng 03] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. "A Neural Probabilistic Language Model". *JOURNAL OF MACHINE LEARNING RESEARCH*, Vol. 3, pp. 1137–1155, 2003.

[Bohn 10] B. Bohnet, L. Wanner, S. Mille, and A. Burga. "Broad Coverage Multilingual Deep Sentence Generation with a Stochastic Multi-Level Realizer". In: *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pp. 98–106, Coling 2010 Organizing Committee, Beijing, China, Aug. 2010.

[Brin 98] S. Brin and L. Page. “The Anatomy of a Large-scale Hypertextual Web Search Engine”. *Comput. Netw. ISDN Syst.*, Vol. 30, No. 1-7, pp. 107–117, Apr. 1998.

[Budu 17] N. Buduma. *Fundamentals of Deep LearningDesigning Next-Generation Machine Intelligence Algorithms.* 2017.

[Cao 19] M. Cao and J. C. K. Cheung. “Referring Expression Generation Using Entity Profiles”. *School of Computer Science, McGill University, Montreal, QC, Canada MILA, Montreal, QC, Canada*, 2019.

[Chen 00] Cheng, Hua and Mellish, Chris. “Capturing the interaction between aggregation and text planning in two generation systems”. In: *Proceedings of First International Conference on Natural Language Generation (INLG’00)*, pp. 186–193, Mitzpe Ramon, Israel, 2000.

[Chen 96] S. F. Chen and J. Goodman. “An Empirical Study of Smoothing Techniques for Language Modeling”. In: *34th Annual Meeting of the Association for Computational Linguistics*, pp. 310–318, Association for Computational Linguistics, Santa Cruz, California, USA, June 1996.

[Chol 18] F. Chollet. *Deep Learning with Python.* 2018.

[Clar 96] H. H. Clark. *Using Language.* Cambridge University Press, 1996.

[Dali 99] H. Dalianis. “Aggregation in Natural Language Generation. Computa- tional Intelligence”. 1999.

[Das 07] D. Das and A. F. T. Martins. “A Survey on Automatic Text Summarization”. 2007.

[Deer 90] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman. “Indexing by latent semantic analysis”. *JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE*, Vol. 41, No. 6, pp. 391–407, 1990.

[DeJo 82] G. DeJong. “An Overview of the FRUMP System”. In: W. Lehnert and M. Ringle, Eds., *Strategies for Natural Language Processing*, pp. 149–176, Lawrence Erlbaum, 1982.

[Dinu 14] G. Dinu and M. Baroni. “How to make words with vectors: Phrase generation in distributional semantics”. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 624–633, Association for Computational Linguistics, Baltimore, Maryland, June 2014.

[Dong 18] Y. Dong. *A Survey on Neural Network-Based Summarization Methods.* 2018.

[Dunn 93] T. Dunning. “Accurate Methods for the Statistics of Surprise and Coincidence”. *Computational Linguistics*, Vol. 19, No. 1, pp. 61–74, 1993.

[Dzmi 16] K. C. Dzmitry Bahdanau and Y. Bengio. “NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE”. 2016.

[Evan 02] R. Evans, P. Piwek, and L. Cahill. “What is NLG?”. In: *Proceedings of the Second International Conference on Natural Language Generation*, pp. 144–151, Association for Computational Linguistics, 2002. Creative Commons Attribution Share-Alike License.

[Fike 71] R. E. Fikes and N. J. Nilsson. “STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving”. *Artificial Intelligence*, Vol. 2, pp. 189–208, 1971.

[Gatt 18] A. Gatt and E. Krahmer. “Survey of the State of the Art in Natural language Generation: Core tasks, applications and evaluation”. *Journal of Artificial Intelligence Research*, Vol. 61, No. 1, pp. 65–170, 2018.

[Haoy 19] Z. Haoyu, Y. Gong, Y. Yan, N. Duan, J. Xu, J. Wang, M. Gong, and M. Zhou. “Pretraining-Based Natural Language Generation for Text Summarization”. p. , 02 2019.

[Hoch 91] S. Hochreiter. “Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München”. 1991.

[Hoch 97] S. Hochreiter and J. Schmidhuber. “Long short-term memory”. *Neural computation*, Vol. 9, No. 8, pp. 1735–1780, 1997.

[Jana 14] S. Janarthanam and O. Lemon. “Adaptive Generation in Dialogue Systems Using Dynamic User Modeling”. *Computational Linguistics*, Vol. 40, No. 4, pp. 883–920, Dec. 2014.

[Jean 15] S. Jean, K. Cho, R. Memisevic, and Y. Bengio. “On Using Very Large Target Vocabulary for Neural Machine Translation”. In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 1–10, Association for Computational Linguistics, Beijing, China, July 2015.

[Jeka 17] O. D. Jekaterina Novikova and V. Rieser. “The E2E Dataset: New Challenges For End-to-End Generation”. 2017.

[Jone 98] K. S. Jones. “Automatic Summarising: Factors and Directions”. In: *Advances in Automatic Text Summarization*, pp. 1–12, MIT Press, 1998.

[Kons 13] I. Konstas and M. Lapata. “A Global Model for Concept-to-Text Generation”. *Journal of Artificial Intelligence Research*, Vol. 48, pp. 305–346, 2013.

[Kost 19] S. Kostadinov. “Understanding Encoder-Decoder Sequence to Sequence Model”. 05 2019.

[Kriz] A. Krizhevsky, V. Nair, and G. Hinton. “CIFAR-10 (Canadian Institute for Advanced Research)”.

[Lapa 06] M. Lapata. “Automatic Evaluation of Information Ordering: Kendall’s Tau”. *Computational Linguistics*, Vol. 32, No. 4, pp. 471–484, 2006.

[Lin 04] C.-Y. Lin. “ROUGE: A Package for Automatic Evaluation of Summaries”. In: *Text Summarization Branches Out*, pp. 74–81, Association for Computational Linguistics, Barcelona, Spain, July 2004.

[Liu 19] Y. Liu and M. Lapata. “Text Summarization with Pretrained Encoders”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3730–3740, Association for Computational Linguistics, Hong Kong, China, Nov. 2019.

[Luhn 58] H. P. Luhn. “The Automatic Creation of Literature Abstracts”. *IBM Journal of Research and Development*, Vol. 2, No. 2, pp. 159–165, 1958.

[Löh 19] T. Löhr and T. Bohnstedt. “Image Classification on the CIFAR10 Dataset”. 2019.

[M Th 01] d. P. J.-R. e. K. M. Theune, E. Klabbers and J. Odijk. “From data to speech: a general approach”. *Natural Language Engineering*, 2001.

[Mani 01] I. Mani. *Automatic Summarization*. Vol. 3, 01 2001.

[Mani 16] E. Manishina. “Data-driven natural language generation using statistical machine translation and discriminative learning”. *Computation and Language [cs.CL]. Université d’Avignon*, 2016.

[Mani 99a] I. Mani and M. Maybury. “Advances in Automatic Text Summarization”. pp. 123–136, The MIT Press, 1999.

[Mani 99b] I. Mani. “Advances in Automatic Text Summarization (The MIT Press)”. MIT Press, 1999.

[Marc 00] D. Marcu. *The Theory and Practice of Discourse Parsing and Summarization.* 01 2000.

[McCu 43] W. S. McCulloch and W. Pitts. "A logical calculus of the ideas immanent in nervous activity". *The bulletin of mathematical biophysics*, Vol. 5, No. 4, pp. 115–133, 1943.

[McDo 93] D. D. McDonald. "Issues in the choice of a source for Natural Language Generation". *Computational Linguistics*, Vol. 19, No. 1, pp. 191–197, 1993.

[Mehd 17] M. A. S. S. E. D. T. J. B. G. K. K. Mehdi Allahyari, Seyedamin Pouriyeh. "Text Summarization Techniques: A Brief Survey". *Computation and Lignuistics*, 2017.

[Minh 15] C. D. M. Minh-Thang Luon, g Hieu Pham. "Effective Approaches to Attention-based Neural Machine Translation". 2015.

[Mitc 12] M. Mitchell, J. Dodge, A. Goyal, K. Yamaguchi, K. Stratos, X. Han, A. Mensch, A. Berg, T. Berg, and H. Daumé III. "Midge: Generating Image Descriptions From Computer Vision Detections". In: *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 747–756, Association for Computational Linguistics, Avignon, France, Apr. 2012.

[Muga 18] J. Mugan. "Generating Natural-Language Text with Neural Networks". 07 2018.

[Nenk 04] A. Nenkova and R. Passonneau. "Evaluating Content Selection in Summarization: The Pyramid Method". In: *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*, pp. 145–152, Association for Computational Linguistics, Boston, Massachusetts, USA, May 2 - May 7 2004.

[Olah 15] C. Olah. "Understanding LSTM Networks". 08 2015.

[Powe 14] Power and Williams. "Generating numerical approximations". *Computational Linguistics*, 2014.

[Rade 02] D. R. Radev, E. Hovy, and K. McKeown. "Introduction to the Special Issue on Summarization". *Computational Linguistics*, Vol. 28, No. 4, pp. 399–408, 2002.

[Rade 98] D. R. Radev and K. R. McKeown. "Generating Natural Language Summaries from Multiple On-Line Sources". *Computational Linguistics*, Vol. 24, No. 3, pp. 469–500, 1998.

[Ramo] J. Ramos. "Using TF-IDF to Determine Word Relevance in Document Queries".

[Reit 00] E. Reiter and R. Dale. "Building applied Natural Language Generation System". No. 1, 2000.

[Reit 97] E. Reiter and R. Dale. “Building applied natural language generation systems”. *Natural Language Engineering*, Vol. 3, No. 1, pp. 57–87, 1997.

[Ries 11] V. Rieser and O. Lemon. *Reinforcement learning for adaptive dialogue systems: a data-driven methodology for dialogue management and natural language generation. Theory and Applications of Natural Language Processing*, Springer, 2011.

[Scie 15] G. C. S. R. Scientist. “Computer, respond to this email.”. *Google AI Blog*, 11 2015.

[See 17] A. See, P. J. Liu, and C. D. Manning. “Get To The Point: Summarization with Pointer-Generator Networks”. In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1073–1083, Association for Computational Linguistics, Vancouver, Canada, July 2017.

[Sing 17] P. Singh. “Neuron explained using simple algebra”. 2017.

[Stei 09] J. Steinberger and K. Jezek. “Evaluation Measures for Text Summarization.”. *Computing and Informatics*, Vol. 28, No. 2, pp. 251–275, 2009.

[Suts 14] I. Sutskever, O. Vinyals, and Q. V. Le. “Sequence to Sequence Learning with Neural Networks”. In: Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., *Advances in Neural Information Processing Systems 27*, pp. 3104–3112, Curran Associates, Inc., 2014.

[Swan 14] B. Swanson, E. Yamangil, and E. Charniak. “Natural Language Generation with Vocabulary Constraints”. In: *Proceedings of the Ninth Workshop on Innovative Use of NLP for Building Educational Applications*, pp. 124–133, Association for Computational Linguistics, Baltimore, Maryland, June 2014.

[Thom 77] H. Thompson. “Strategy and Tactics: a Model for Language Produc- tion”. *Papers from the 13th Regional Meeting of the Chicago Linguistic Society*, 1977.

[Torr 14] J.-M. Torres-Moreno. “Automatic Text Summarization”. *Wiley*, 2014.

[Vasw 17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. “Attention is All you Need”. In: I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., *Advances in Neural Information Processing Systems 30*, pp. 5998–6008, Curran Associates, Inc., 2017.

[Wang 09] T. Wang and G. Hirst. “Extracting Synonyms from Dictionary Definitions”. In: *Proceedings of the International Conference RANLP-2009*, pp. 471–477, Association for Computational Linguistics, Borovets, Bulgaria, Sep. 2009.

[Whit 15] M. White and D. M. Howcroft. "Inducing Clause-Combining Rules: A Case Study with the SPaRKy Restaurant Corpus". In: *Proceedings of the 15th European Workshop on Natural Language Generation (ENLG)*, pp. 28–37, Association for Computational Linguistics, Brighton, UK, Sep. 2015.

[Xie 17] Z. Xie. "Neural Text Generation: A Practical Guide". 2017.

[Yann 98] L. B. Yann LeCun, Patrick Haffner and Y. Bengio. "Object Recognition with Gradient-Based Learning". 1998.