TECHNISCHE HOCHSCHULE NÜRNBERG
GEORG SIMON OHM

Faculty of Computer Science

IT-based Text Generation with the use of NLP 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	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 übertragba-rem 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 an massive amount of texts and documents. Every day there are new articles, news, documentations and reports packed with information. For this reason, a new discipline, especially for students like me arose out of this. Knowledge is nowadays accessible everywhere and immediately, but the consumption takes way too much time. Websites like *https://www.blinkist.com/de* provide their costumers text summarizations of different kind of books readable in 15-30 minutes. This is an interesting way to safe time, but still this summarizations 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 bad and no human wants to read a grammatically broken summarizations. Bt Deep Learning changed the game completely, because algorithms are now feasible enough to summarize texts as good as humans.

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.

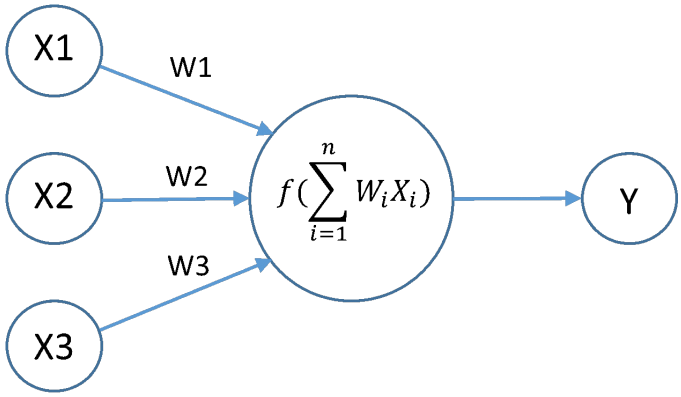

Figure 1.1: A simple Neuron with 3 inputs and 1 output [Sing 17]

To illustrate the basic workflow of a text summarizing system, I programmed a prototype. 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 the user and costumer experience. The phrase *Machine Learning* was originally 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 the neural communication. Figure 1.1 illustrates a mathematical approximation of the humans brain cell in form of an *artifical* neuron. Nowadays, this brain-neuron based model is mostly declared to be not realistic enough [Andrew Ng, deeplearning.ai], because the structure of a brains neuron is far more complex than the illustration in figure 1.1 suggests. Nevertheless, it provides a really good 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 effective and efficient than previously used algorithms. Many frequently held conferences around the globe proof 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 a 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, whereas 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 10 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 precise 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 actually 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 very broad term and can be applied on 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 researches define it very differently. The core can be broken down to, Deep Learning allows to build 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.

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

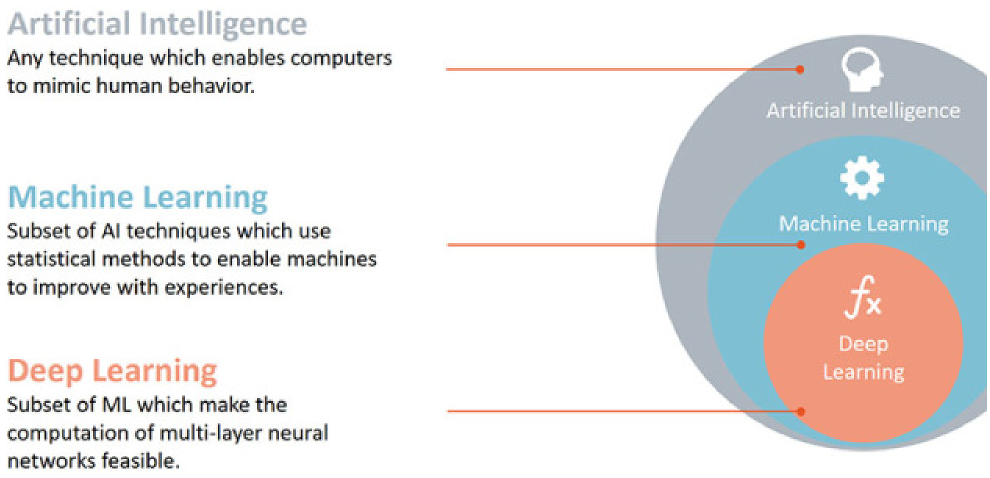

Figure 1.2: Zoom into Artificial Intelligence from *https://rapidminer.com/blog/artificial-intelligence-machine-learning-deep-learning/*

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 different sources, e.g. images, videos or speech,

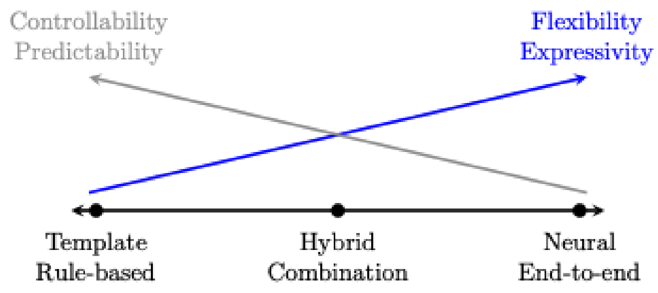

Figure 1.3: Rule-Based vs. Neural-Text-Generations System [Xie 17], Page 4

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. Text generation is therefore in general the output part of an input-output NLP system.

But what defines a summarization? Literature points out multiple different definitions. One definition proposes that the summary of a document is the process of distilling the most important 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 in general 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 major disadvantage is that the necessary computing power has increased exponentially. 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*. But 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 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
- 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. Further explanation in Section 2.2.2.1 of Chapter 2.

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

As a human, creating an 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 texts essence in such a way, that it shortens the original document to a minimum down. However, after a period of time the person is most likely to summarize the exact same text in a different way than for example 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 in a different way, into different fragments with multiple relevance for each fragment. A crucial role is also the coherence of a text. Different applications of text summarizations are:

- Multi-Document Summarizations
- 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].

As an illustration and example for my entire thesis, besides of my own prototype (Chapter 3), I start with a case study for the entire Chapter 2.

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 the 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 good as possible. The search engine Google is known for its accurate any 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]. The achieve 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, for a good result the model who generates the headlines only needs the first few lines of an article.

All examples from this thesis will be conducted from the perspective of an 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 major 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 out 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 personal 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 computional high demanding power. I am going to mention which parts are the fundamentals for my prototype and which technologies are state of the art, but not possible to achieve with a basic home computer.

2.1 The Structure of Text Generation

Text Generation, Language modeling or Natural Language Generation are different words for basically the same meaning, but I will keep the denotation of Text Generation. 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 understandable 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 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 can be not 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 subproblems [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

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

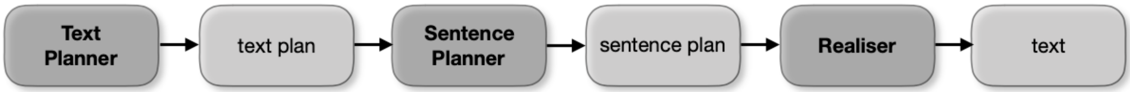

Figure 2.1: Classical 3-stage Text Generation architecture, after Reiter and Dale (2000) [Reit 00]

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

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

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.

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

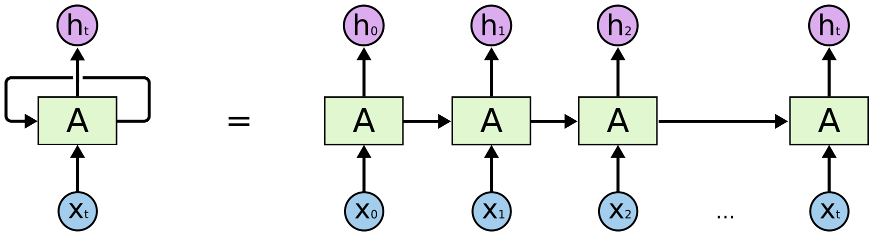

Figure 2.3: Recurrent Neural Network with integrated loops [Olah 15]

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 information. The goal of the cell is to solve the vanishing gradient problem of the Recurrent Neural Network. Inputs into this cell

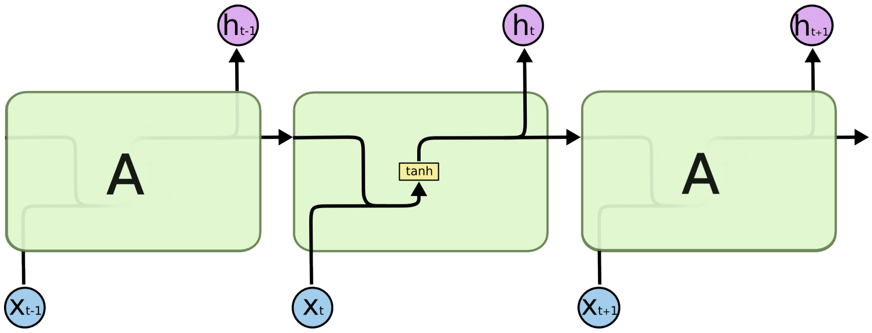

Figure 2.4: The repeating module in an Recurrent Neural Network contains one single layer [Olah 15]

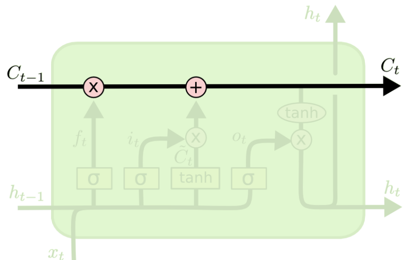

Figure 2.5: Cell State of the Long Short Term Memory which acts as data highway [Olah 15]

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
- 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.

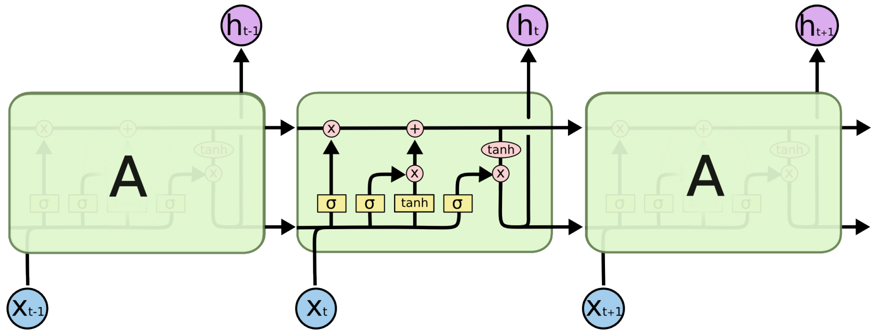

Figure 2.6: The repeating module in an LSTM contains four interacting layers [Olah 15]

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 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.

2.2.2.1 N-Grams

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 p($g|e$) is trained on multiple texts (corpus) and finally outputs 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

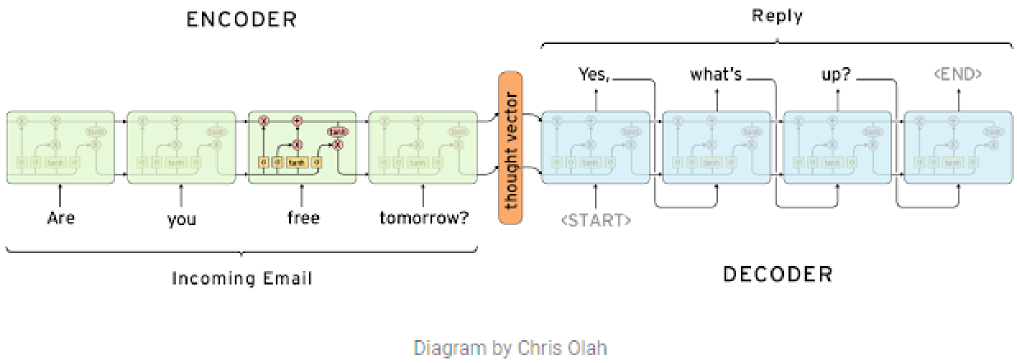

Figure 2.7: LSTM encoder-decoder model for automated E-Mail reply

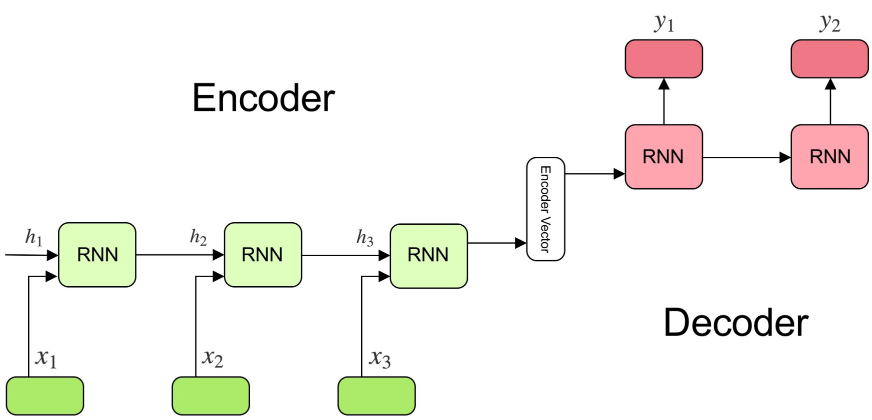

Figure 2.8: Encoder-decoder sequence to sequence model [Kost 19]

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 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.

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]

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, 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 2.11) 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:

"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. The Long Short Term Memory is not starting to right before time step $t = 6$ at state h_6. 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. Attention also projects its vectors into the meaning space (Figure 2.11.

Sequence to sequence models can be entirely built up from the attention model [Vasw 17].

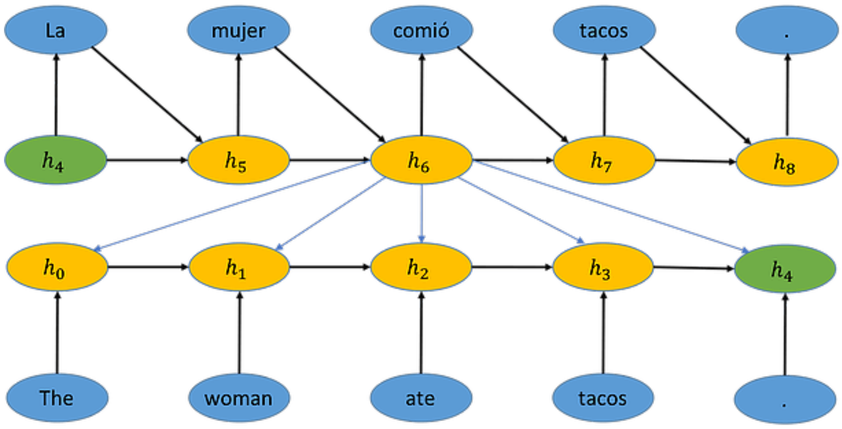

Figure 2.10: Attention mechanism for Spanish-English translation [Muga 18]

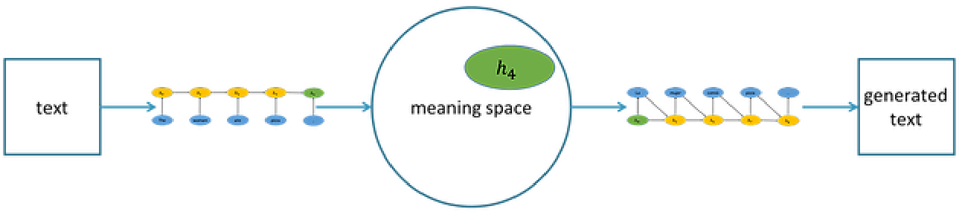

Figure 2.11: Meaning Space of the Attention model [Muga 18]

2.3 The Structure of Text Summarization

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.

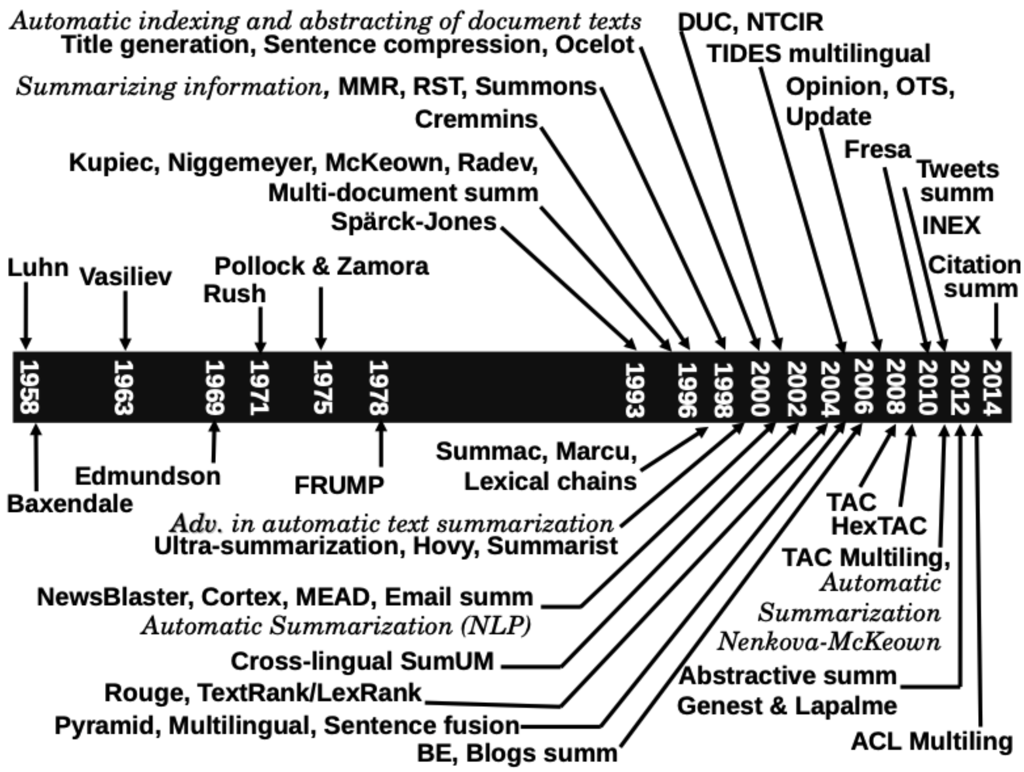

Figure 2.12: Highlights of automatic text summarization [Torr 14] (Page 17)

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 2.12, I think it provides an good overview on the development of automatic text summarization. Everything started with Peter Luhn from Germany in 1958.

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**.

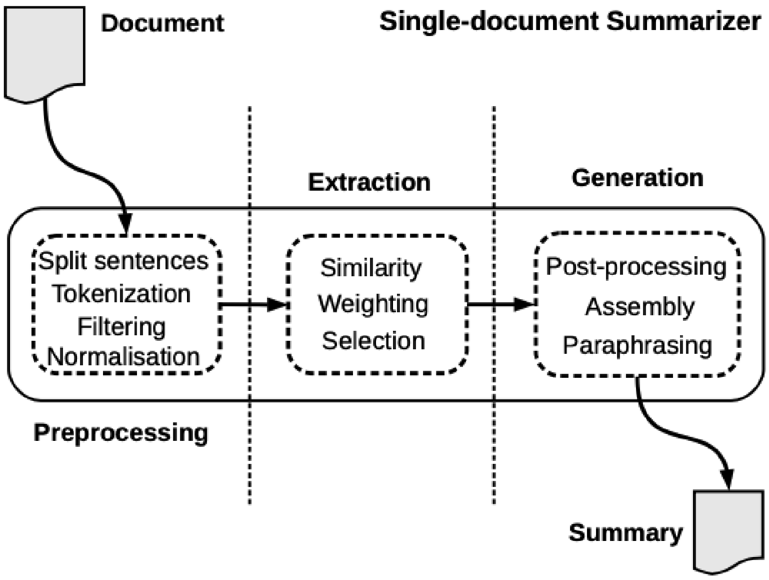

Figure 2.13: General architecture of a extraction-based single-document summarization system [Torr 14] (Page 70)

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 2.13 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 PAGERANK 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 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 2.14.

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

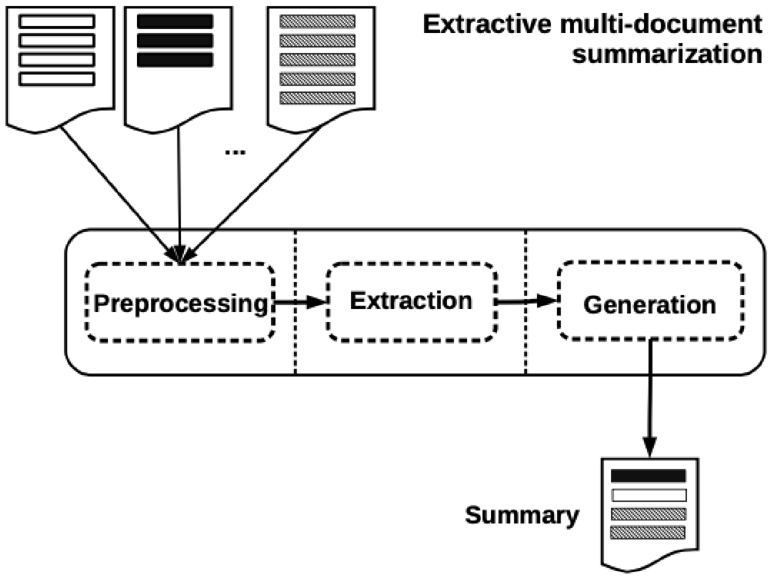

Figure 2.14: Extraction based multidocument summarization system [Torr 14] (Page 110)

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 2.14). 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 2.14. 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

Informative vs Indicative

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.

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.

2.3.3 Output

2.3.3.1 Extractive

Figure 2.16 shows a simplified process of the abstractive or extractive approach. Both approaches can be applied decoupled from each other. The source documents, either single document or multi document go in as well with the topic of the topic (scientific, article, ...). The parameters are the compression rate τ, the type, etc.

2.3.3.2 Abstractive

2.3.4 Evaluation

ROGUE BLEU

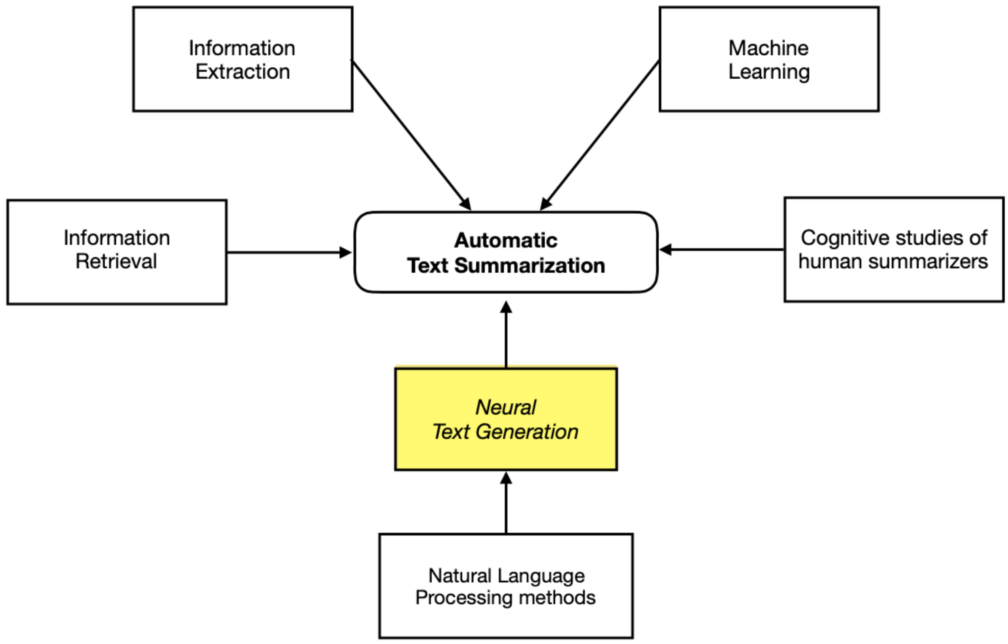

Figure 2.15: Research fields with influence on the development of text summarization

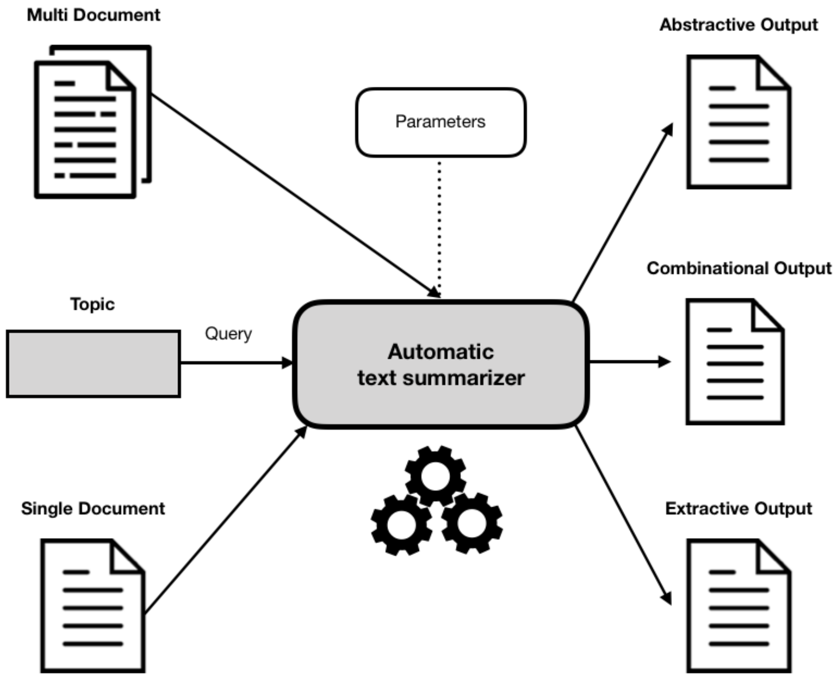

Figure 2.16: Simplified Abstraction Extraction process

2.4 Advanced Approaches for Text Summarization

2.4.1 Neural Text Summarization

2.4.2 Combinational Approach

2.4.3 Reinforcement Learning

Chapter 3

Prototype

My prototype shall do this: ... , because ...

3.1 Objective

Textsummarization

3.2 Technical concept

Fachkonzept - Proto

3.2.1 Structure

The different steps of Text Generation

- Importing Dependencies
- Loading the Data
- Creating Character/Word mappings
- Data Preprocessing
- Modelling
- Generating text

3.2.2 Neuronal Net

LSTM

RNN

3.2.3 Process Modeling

NLP Bilder

3.2.4 Data flow modelling

Attention summarization

+ NLP Bilder

3.3 Implementation

Code for the Machine Translating

3.4 Evaluation

Print Ergebnisse

Bild

Image Caption

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.

[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.

[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.

[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.

[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.

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

[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.

[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.

[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.

[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.

[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.

[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.

[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.

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

[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.