TECHNISCHE HOCHSCHULE NÜRNBERG
GEORG SIMON OHM

Faculty of Computer Science

IT-based Automatic Text Summarization with the Use of Textgeneration 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

Abstract

Text Summarization can be a powerful tool to reduce the amount of time for reading documents, articles or even research papers. The thesis is divided into a larger state of the art part and a shorter prototype part. The state of the art part examines the concepts of text generation and text summarization with the focus on my prototype. Most concepts are introduced in order to fully understand how my prototype is able to achieve the text generation, except for some advanced thinking outside the box concepts, which cannot be applied by me, because it would exceed this thesis. My prototype is trained on the Amazon-fine-food-reviews from *www.kaggle.com* and the results are evaluated on the Rouge and BLEU scores. In the end, I further introduce some performance enhancements.

Acknowledgement

I would like to thank Professor Dr. Albrecht Holl for supervising my Bachelor Thesis. He always supported me and helped me a lot when I was facing problems regarding my thesis.

I dedicate my thesis to Wong Chui Shan (黃翠珊)

Preface

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 primary interest among all IT related topics is artificial intelligence.

My 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 *text summarization* within NLP. I research about the history of text summarization systems and the current state of the art. Further I program a prototype in python to apply this knowledge.

For my research, I encountered a lot of old and recently published papers, mostly from *https://arxiv.org/*. Reading through the papers requires much prior knowledge, especially in mathematics, which I learned during my semester in Hong Kong at the City University of Hong Kong. Machine Learning and, more specifically, NLP is not an intuitive study. All of my explanations could be presented with their mathematical notations as well, but it would exceed this thesis by far if I would explain every technological change based on its maths. I rather want to give a detailed overview and if the reader is encouraged to learn more about it, he/she knows then the keywords to search for it and all of the papers are cited as well. During the five-month development process of the bachelor thesis, I gained much knowledge. I recognized that NLP is a vast topic, always 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* text generation related disciplines is not possible within this thesis. For that reason, I have primarily focused on text summarization part.

Chapter		Section	From where? Knowledge-Input	From where? Question-Input	How? Method	What? Target-Description
0	**IT-based Automatic Text Summarization with the use of Text Generation Methods** State of the Art and design of a prototype		• Research Papers • Books • HongKong, TH-OHM • Online Course	1. What is the State of the Art of Text Generators and Text Summarizers? 2. In what quality can I program the text generation prototype myself and what quality does it have compared to other ones?	1. Presentation of the state of the art of Text Generators and Text Summarizers. 2. Study of the relevant aspects of Text Generation and programming of an IT-based Text Summarizer prototype.	1. State of the art technical elaboration. 2. Programming of a prototypical algorithm that generates a summary for a given review input
1	Introduction	1.1	Structure of this Thesis	• [0.1] • Scientific Writing	1. How is my thesis structured? 2. Which questions is my thesis going to answer, why should someone read it?	1. Brief overview over the thesis 2. Short but detailed introduction to all Chapters of this thesis.	1. Pointing out the fundamental points 2. Answer to the question, why my bachelor thesis makes sense and what is my motivation for
		1.2	Machine Learning	• Hong Kong ML course	1. How is Text Summarization related to Machine Learning?	1. Presentation of all relevant aspects that belong to Text Summarization	1. Zoom-In introduction from Artificial Intelligence into Text Summarization
		1.3	Case study	• Online Research	1. What are current useful applications of word processing systems?	1. Research on the current and planned Text Summarizer, in the field of word processing.	1. Explanation through an interesting easy introduction.
2	Evolutionary View on the State of the Art	2.1	Text Generation Concepts	• [1] • Paper Research	1. Which historical achievements are necessary to know for understanding the advanced concepts of Text Generation?	1. Research on the relevant aspects of state of the art research chronologically at different time steps for this topic.	1. Description of the application-oriented models for this topics using formulas and explanations.
		2.2	Advanced Approaches for Text Generation	• [1] • [2.1] • Books	1. What are current Text Summarizer systems capable of? 2. Where are current application fields?	1. Literature research of the Text Generation history (~60 years). 2. Literature research of current papers	1. Presentation of the history of Text Generation in the form of a chronological sequence. 2. Use of the first technologies
		2.3	Text Summarization Concepts	• [1] • Paper Research	1. Which historical achievements are necessary to know for understanding the advanced concepts of Text Generation?	1. Research on the relevant aspects of state of the art research chronologically at different time steps for this topic.	1. Description of the application-oriented models for this topics using formulas and explanations.
		2.4	Advanced Approaches for Text Summary	• [1], [[2.1], 2.2], [2.3] • Books	1. What are current Text Summarizer systems capable of? 2. Where are current application fields?	1. Literature research of the Text Generation history (~40 years). 2. Literature research of current papers	1. Presentation of the history of Text Summary in the form of a chronological sequence. 2. Use of the first technologies
3	Prototype	3.1	Objective	• [0] • [1] • [2]	1. What should my prototype be able to achieve with given resources? 2. Which output can be expected in the best case?	1. Reverse Engineering 2. Classification and analysis of possible results,e.g. whether the output is grammatically correct.	1. Explanation of the scope of my prototype. 2. Collection and classification of requirements for the algorithm and its output.
		3.2	Technical concept	• [2] • [3.1] • Online Research	1. How is my prototype structured? 2. Which algorithms do I use? 3. Which processes does the data go through? 4. How is the data processed?	1. Creation of a technical concept 2. Modeling Algorithm 3. Process Modeling 4. Data flow modeling and, or data modeling	1. Technical concept completed. 2. The prototype is modelled without IT reference on the basis of various submodels. 3. The individual processes are modelled without a concrete implementation proposal. 4. Data processing visualized
		3.3	Implementation	• [3.1] • [3.2] • Online Courses	1. Which technologies do I use for my prototype: - "What Python libraries and IDE?" - "What are the HW & SW requirements?"	1. Create software dependency portfolio - research of required libraries - research of required hardware, software and selection	1. Creation of an IT concept in the form of a description of the necessary technical means using sub-models
		3.4	Evaluation	• [3.1] • [3.2] • [3.3] • Paper Research	1. How is the output of the prototype evaluated, e.g. with which metrics? 2. How to evaluate the quality of the output? 3. How can the algorithm be improved?	1. Target/actual comparison of the requirements with the output of the prototype. 2. Comparison with related work and metrics. 3. Research on potential improvements of the algorithm.	1. Evaluation and analysis of the result based on grammatical correctness and meaning. 2. Compare better results with my result. 3. Evaluate possible optimization techniques (Chapter 2) for my prototype.
4	Transferable knowledge	4.1		[0] to [3]	By which elements could my project be modularly extended to produce a different or better result?	Generalization from the results obtained so far.	Placing the evaluation results in a social context.

Contents

Chapter 1.

Introduction

Why do we do basic research? To learn about ourselves.

Walter Gilbert

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 most people. Deep Learning changed the game entirely, because the state of the art algorithms are now feasible enough to summarize texts as good as humans do.

1.1. Structure of this Thesis

My thesis aim 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 significant impact on the readability for humans. For readers who are not familiar with machine learning in general, I will provide a zoom-in introduction from artificial intelligence in a broader term into the tiny subfield of 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

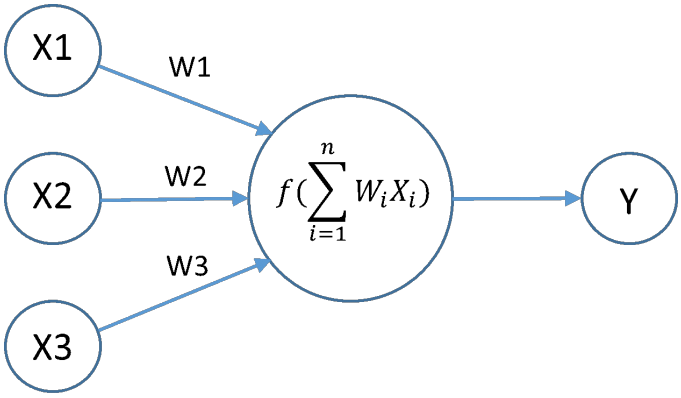


Figure 1.1.: A simple Neuron with three inputs and one output [Sing 17]

sometimes usable vice-versa. For this reason, I provide the most crucial historical achievements for text generation in combination with the latest text summarization results, because both topics intersect in many aspects. The fundamental concept of historical and modern approaches to summarize and generate text is 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. This concept, development and the evaluation are located in chapter 3 - Prototype, but it requires prior knowledge to understand the mechanism from the input to the output fully. Finally, in the last chapter I will discuss further improvements for my prototype and a brief discussion of the 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 about 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 300 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.

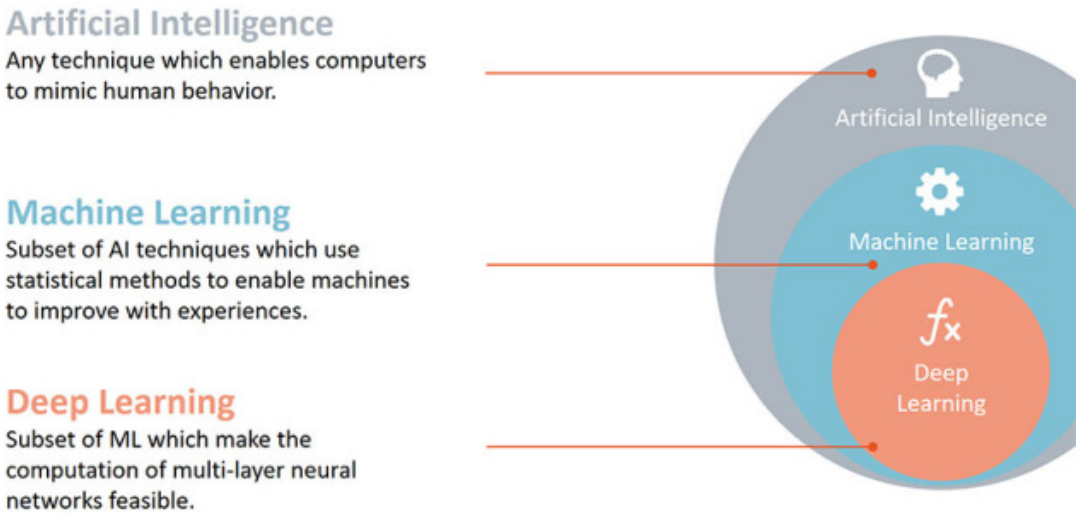


Figure 1.2.: Zoom into Artificial Intelligence from *https://rapidminer.com/blog/artificial-intelligence-machine-learning-deep-learning/*

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. Therefore, text generation is 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 computationally powerful computer and most language models are trained on a huge data-set. For a researcher at home, this can only be achieved through an expensive graphics card (GPU) or renting a server from the *Amazon Web Services*, for example. 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 of the network's complexity it becomes more and more challenging to understand the decisions of the neural network. Neural Networks are still, to a large extent, a *black box*. Nevertheless, especially in NLP, they achieve 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

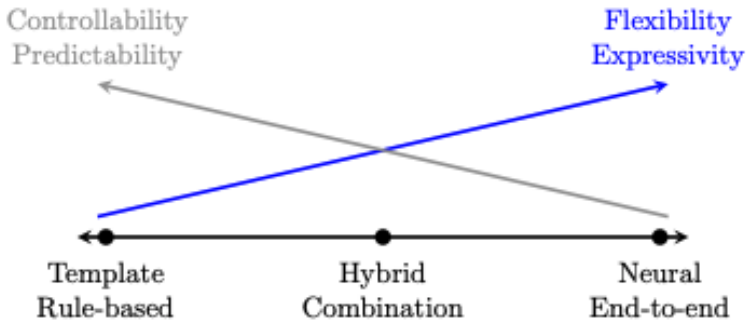


Figure 1.3.: Rule-Based vs. Neural-Text-Generations System [Xie 17], Page 4

- 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 and following 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 relevant 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

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 model's input in form of the first sentence of a news article and the second column shows the model's prediction for the headline [Scie 15].

- Scientific Research Papers
- News Headlines

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. The company 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. The prototype of this thesis will be conducted about food reviews from the company *Amazon*.

Chapter 2.

An Evolutionary View on the State of the Art

Each problem that I solved became a rule, which served afterwards to solve other problems.

Rene Descartes

The goal of this chapter is to survey the development of the text generation from the very beginning until today, 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 generic 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 different 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 demand a powerful graphics unit. Therefore, this chapter is 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 to achieve for big companies like Google.

Table in Figure 2.1 illustrates the methods which I am going to use in my prototype in Chapter 3. Most of the Sections and Subsections from this current Chapter will find its way into my prototype, except Section 2.4, because it would exceed the limits of my bachelor thesis. The other Sections discuss different approaches through time, but this table shows exactly which specific mentioned approaches I implement in my prototype.

Methods used in my prototype in Chapter 3			
Section	**Subsection**	**Approach**	**Purpose**
Text Generation Concepts	Architectures and Approaches	**Data-driven**	I have the necessary computing power to apply this advanced method for my prototype
Advanced Approaches for Text Generation	Recurrent Neural Networks	**Not used**	Substituted by the LSTM
	Long Short Term Memory	**Attention**	Necessary for Attention
	Sequence to Sequence	**Attention**	Necessary for Attention
	Encoder and Decoder	**Attention**	Necessary for Attention
	Attention	**Attention**	This approach yields a very accurate output, because it keeps track of the structure of the text
Text Summarization Concepts	Input	**Multi-document**	I have used ~500.000 different reviews stored in different documents
	Purpose	**Headline Summary**	The reviews are compressed into a single review, that is purpose of the headline summary
	Output	**Abstractive approach**	I have the necessary computing power to apply this advanced method for my prototype
Advanced Approaches for Text Summarization		None	I don't implement an advanced approach, because it would the exceed the complexity of this thesis

Figure 2.1.: Tabular overview of methods from the upcoming Chapter 2, which are going to be used in my prototype from Chapter 3. From the Text Generation Concepts, Advanced Approaches for Text Generation and Text Summarization Concepts, I introduce different approaches, but for my prototype, I only use only the text summarization concepts and not the most advanced approaches

2.1. Text Generation Concepts

Text Generation, Language Modeling or Natural Language Generation are different words with similar meanings. 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 (AI) and computational linguistics that is concerned with the construction of computer systems that can produce readable texts in English or other human languages from some underlying non-linguistic representation of information' [Reit 97]. This definition rather implies 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 (NTG), 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 illustrate the first approach for early decision processes. The chronological order plays a vital role for this tasks. Furthermore, they can be distinct into a strategy and tactics part. Thompson H. declared these two distinctions for the first time in 1977 [Thom 77]. Still, when it comes to a 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. No aggregation is necessary for the headline example.

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 particular case, the determined content could be the first three sentences. The process of shortening down a document into a summary, the text's key points must be stored into some collection of preverbal messages. A semantic representation of information is often expressed in the form of a logical or database like style [Gatt 18]. That can be achieved through grouping semantically similar words and phrases and to remove redundancies (e.g., plays and play into only play). 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. These topics will be automatically 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 and order of these fragments need to be determined. 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 the 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 my example, the content was already predefined in the first step, now the important words and sentences will be reordered based on these 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 football games and its results published in 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

The three tickers are not redundant, because they contain 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. White and Howcraft (2015) propose a general approach to this problem. 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. Lexicalization

After the sentences have been aggregated and finalized, the next step is lexicalization, 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 of the lexicalization step correlates with the number of alternative sentences available. Furthermore, it is important if their 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. This process decision needs to be decided in advance. For example, the soccer game is more likely to be converted into different styles than a

weather forecast. Another important difficulty is to design the way in how the lexicalization cares about gradable properties. If the live ticker was:

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

Then the system 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 meanings of words differently, as Power and Williams (2014) pointed out in an evaluation. The time *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 lexicalization, 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 be broken down into two steps. The first step is to decide the shape of a 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)

Assigning words to named entities can even be used on its on to detect organizations, people or other relevant objects in a sentence.

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 realization, 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 a 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:

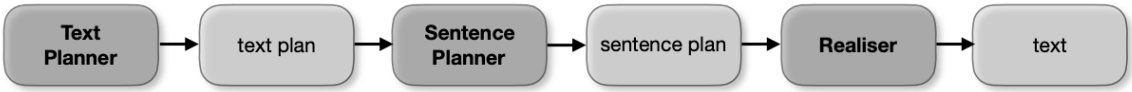


Figure 2.2.: Classical 3-stage Text Generation architecture, after Reiter and Dale (2000) [Reit 00]

- 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 use of the statistics. 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 organized 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. SVM is a machine learning approach, and it tested on a standard 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 complicated it becomes, the more abstract the output will be [**?**] (Page 21). This paves the way for a stochastical end-to-end system like Konstas and Lapata showed in 2013 [Kons 13]. They present a first step towards 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 are organized together. There are three main approaches for the Text Generation architecture shown in the dark boxes in Figure 2.2. The light boxes illustrate the outputs from the main stages.

Since the design of the modules in Figure 2.2, 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 are 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 2.2, 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 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 2.2. 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 several similarities to the *Sentence Planner* from Figure 2.2.

- **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 this phase becomes. It can be seen as the *Realizer* 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, but the abstractive based methods for Text Generation achieve the state of the art results. 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 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 these cases, a rule-based template could make sense, but for the most modern use-cases, this approach is obsolete.

Even in rule-based architectures, there have been many developments proposed, but for the sake of simplicity, I want to discuss the modern approaches in more detail than the 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 the goal to be achieved [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 store actions into tuples containing 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 2.2) [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 usual. This allows the input to be more flexibly processed. The most modern way for a 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 be manually configured. The text generation process could be modeled by a Long Short Term Memory (LSTM) (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 punishment function to adjust the behavior of the desired output.

Rieser et al. (2011) pointed out that this approach is useful in optimizing information presentation when generating restaurant recommendations [Ries 11]. Janarthanam and Lemon (2014) applied this method to improve the choice of information for selecting 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 behavior by changing its internal user-model [Jana 14] (Pages 883-920).

2.1.2.3. Data-driven approach

Data-driven models experience more and more attention recently in the text generation community. They provide the flexibility and potential to overcome the template-based approaches. Even in the past six years, there have been plenty of studies that show the successes of data-driven models. For example, Dinu and Baroni (2014) stated out that text generation can be performed on 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 these methods and data-driven approaches, in general, require much training data. The first step for such a system is to build up a so-called *corpus*. A corpus can be viewed as a kind of template, but the way it is used is entirely different than the template from the rule-based approach. The corpus is created through the training of a vast 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 now contains a set of vocabulary that 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, but mostly the system's response. This can be regarded as a higher abstraction level, used for the modern text summarization systems.

It can be seen in Figure 2.3, that 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 various units as proper names and numerals. The 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.

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 2.3.: Example Corpus from [Mani 16] Page 103

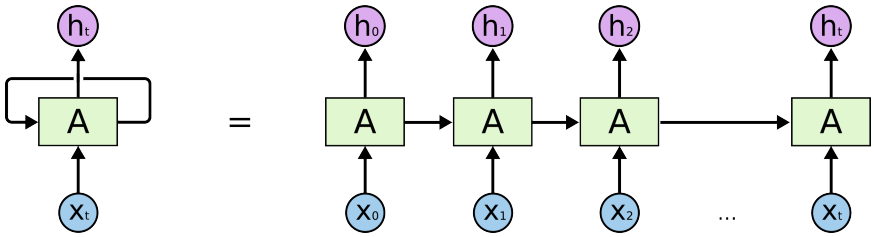


Figure 2.4.: Recurrent Neural Network with integrated loops [Olah 15]

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, a neuron of a basic neural network cannot process time-depending input like language, 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 over time and input new words a sentence into future time steps into the network[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 2.4 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 2.4 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
Time step = t
Input = x
Hidden state = h

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 to the learned probability distribution over a sequence. The output at each time step t is a conditional distribution

$$p(x_t|x_{t-1}, ..., x_1)$$

Where x_t (x on time step t) is dependent on all previous x from time step 1 until *t-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 can 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 can be

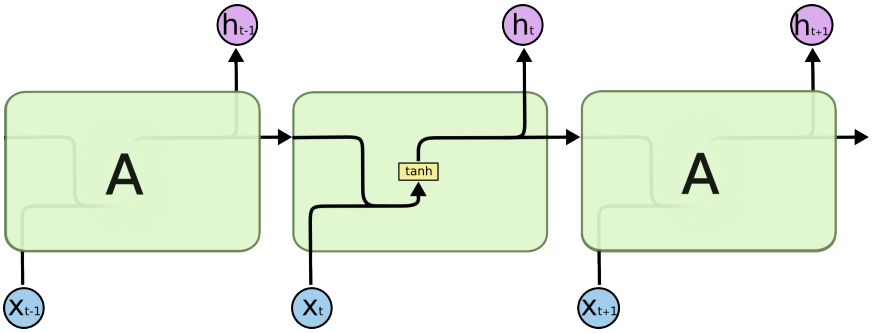


Figure 2.5.: The repeating module in an Recurrent Neural Network contains one single layer [Olah 15]

stored for an extended period, without forgetting them, as in Recurrent Neural Networks. The LSTM is designed to avoid the loss of information (vanishing gradient problem), by intentionally passing on specific information over plenty of time steps.

LSTM's can be enrolled the same way as RNN's, but there is a core difference between the Recurrent Neural Network in Figure 2.5 and the Long Short Term Memory in Figure 2.7.

Figure 2.5 shows three cells of a Recurrent Neural Network, where each cell receives the input x at the time steps *t-1* (past), t (present) and *t+1* (future). Looking at a specific cell, the predecessors cell output is squeezed together with the presents cells input into the **tanh** function to create a so called *activation*. Depending on the two inputs for the **tanh** function, the output of the **tanh** function gets either triggert (*activated*) or not and is saved into the hidden cell at the same time step as the input.

This is a rather simple architecture compared to the Long Short Term Memory cell in Figure 2.7. There is one additional input and output in each LSTM cell, but a lot more additional mathematical operations than in the RNN cell.

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

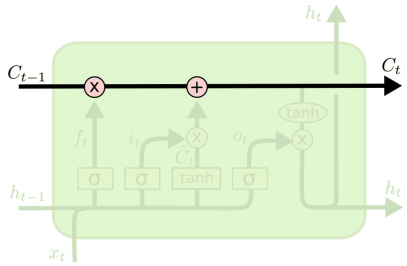


Figure 2.6.: Cell State of the Long Short Term Memory which acts as data highway [Olah 15]

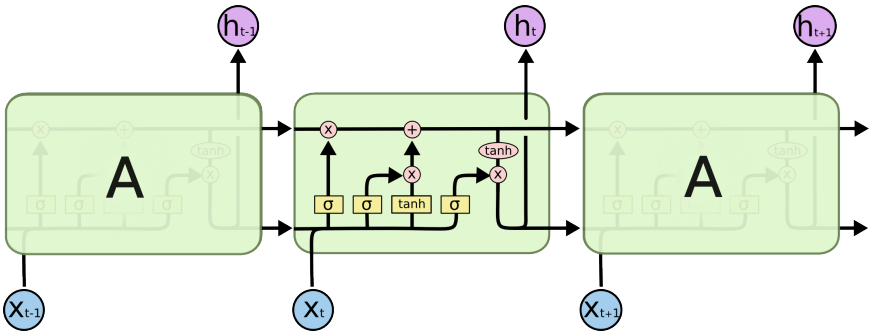


Figure 2.7.: The repeating module in an LSTM contains four interacting layers [Olah 15]

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 2.6. The cell state acts like the information highway in the network. With only some minor linear computation, it runs through the entire cell. This way, information can pass very easily through the entire network.

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 2.4. 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 p($g|e$) is trained on multiple texts (corpus) and finally outputs p(e), which is calculated only on the target corpus in English.

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

Where
g = a german source text
e = the english desired output text
$p(e|g)$ = the conditional distribution
$argmax$ = selecting the result with the highest probability

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 the form of feed-forward fully-connected neural networks produced such excellent 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.

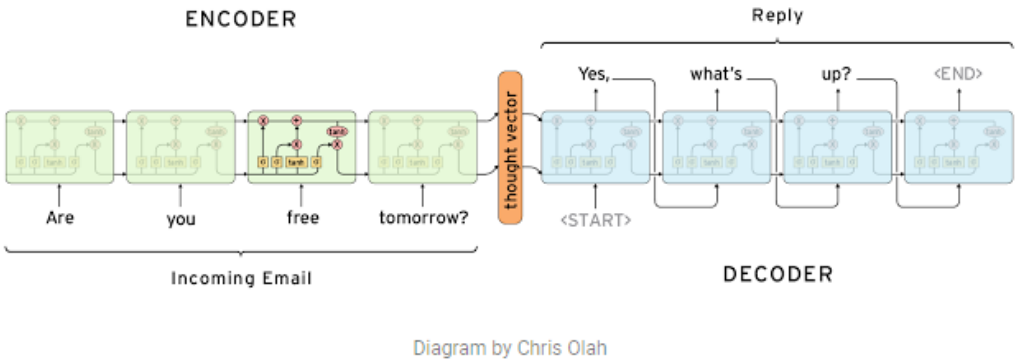


Figure 2.8.: LSTM encoder-decoder model for automated E-Mail reply

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

g = Er rennt
e = He is running

if the source g is translated into English e, the length of the source and output is different. The LSTM itself can not translate that, because of the different word length. The Long Short Term Memory cell from Section 2.2.2 was invented independently from the sequence to sequence models, but finally, three employees of Google published a paper about their approach to making use of the LSTM to create a sequence to sequence model, also called the 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 standard Feed Forward Neural Network, the output of a hidden layer in the network can not be manipulated. The first 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 2.8 illustrates the model for Google's automated email reply.

Figure 2.8 makes use of a 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].

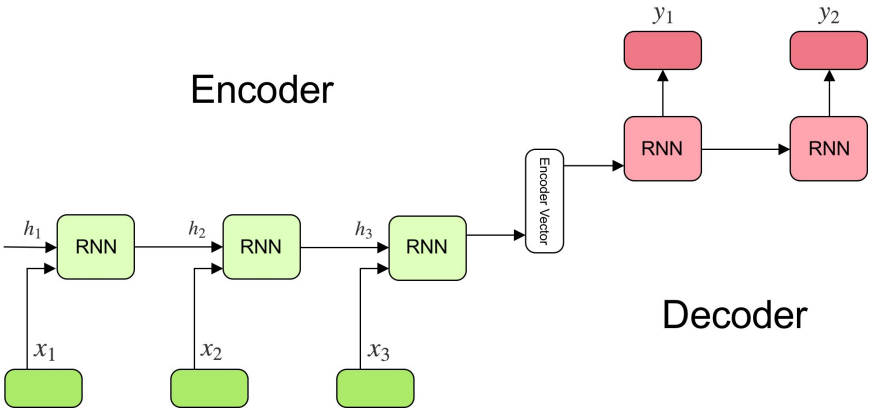


Figure 2.9.: Encoder-decoder sequence to sequence model [Kost 19]

2.2.4. Encoder and Decoder

In the following, the encoder and then the decoder will be explained to have a better insight into how these models work. My prototype from Chapter 3 is based on this kind of model. As already mentioned, a sequence to sequence model is often referred to as an 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.

Figure 2.9 shows that the encoder-decoder structure consists of three parts:

- Encoder
- Intermediate (encoder) Vector
- Decoder

The **Encoder** iteratively integrates the words from a sentence into the hidden state h and further into the Long Short Term Memory cell. Figure 2.6 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*. The cell must compute both the input words, but also the knowledge from the 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 2.10 is [50000 x 300].

A connection of multiple recurrent units (three in Figure 2.9) where each accepts 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 2.9, the hidden state of *h3* is calculated based on the prior two cells.

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 2.10.: Snippet of an example vocabulary table. This table shows actually 50000 words where aardvark is the first and zoology is at position 49999. To represent the words in a vectorspace (meaningspace), I use the artificially chosen dimension size of 300. For this reason the table shown has a size of [50000, 300][Muga 18]

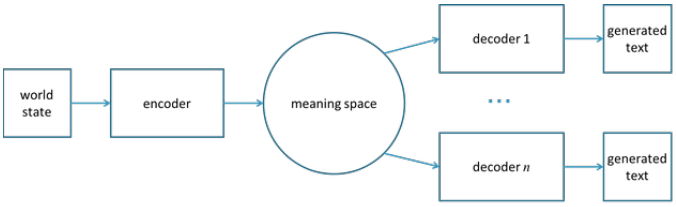


Figure 2.11.: Meaningspace of neural text generation [Muga 18]

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. 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 world state in this figure represents a sentence like in the box below *"Since I am living in Hong Kong, by now, I speak fluently..."*. The encoder is a neural network that maps this sentence into a vector. This is necessary, because the computer doesn't under words, but only numbers. For this reason the words must be transformed into meaningful number representation as shown in Figure 2.10. 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.

Example encoder vector of the word *Cantonese*:

[2.34, 2.15, 3.45, 2.32, ..., 1.36]

where this vector has the shape [1 x 300]

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 squeezes all outputs into a summed up probability of 1. For example:

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

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

The chosen word is **Cantonese** because it has 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 fundamental 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 also multiple vectors where each is capable of capturing other information from the input.

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 2.12 shows the basic concept of attention for the summarization task of a news article. The attention model tries to identify relevant words and meanings in the source text

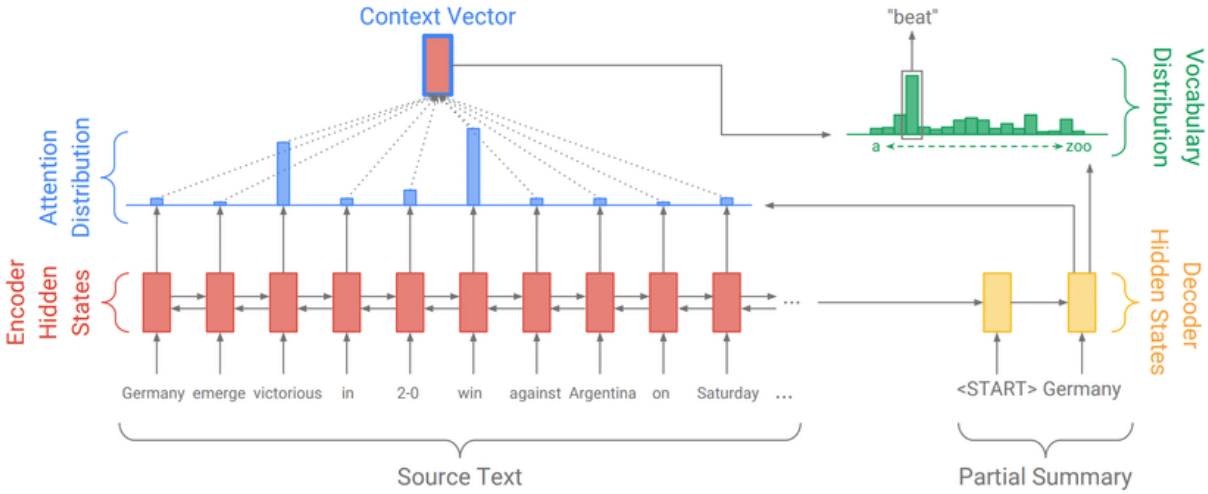


Figure 2.12.: Baseline sequence-to-sequence model with attention [See 17]

to generate new words. This means generating words that 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 essential 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 a data-to-text method, because it takes multiple inputs into account, such as the text, opinion or another vocabulary.

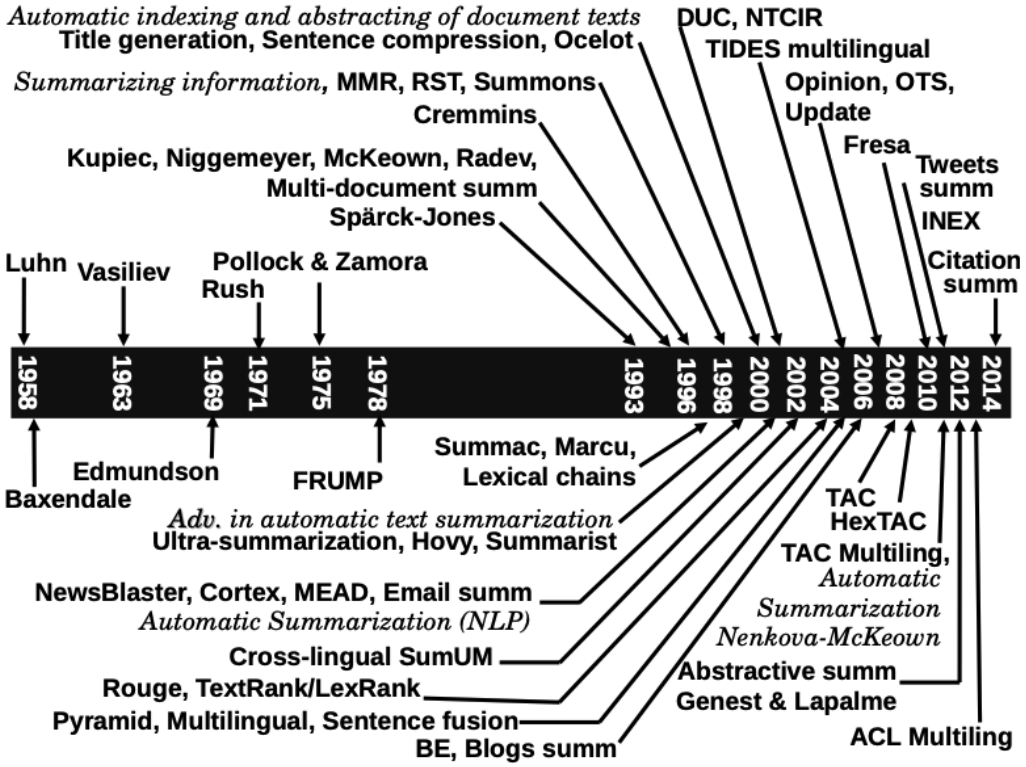


Figure 2.13.: Highlights of automatic text summarization [Torr 14] (Page 17)

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

These three factors are explained in the following.

Figure 2.13 provides a broad overview of 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**.

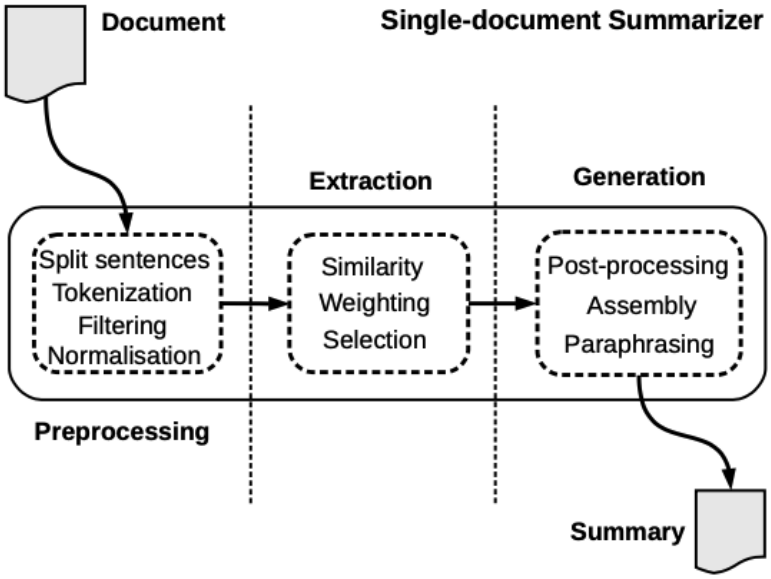


Figure 2.14.: General architecture of a extraction-based single-document summarization system [Torr 14] (Page 70)

2.3.1.1. Single-Document

The most simple task for an automized summarization system is a generic single document summary. Peter Luhn's work first introduced that system in 1958, an extractive method to summarize a text [Luhn 58]. Even though this first approach was proposed over 60 years ago, it is still up to date. 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 2.14 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 that 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). Therefore, this technique measure 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.

S = Matrix

Graph-based approaches conduct to represent the 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 word and collections. Another widely used approach is *bag-of-word*. I used that as an example in Section 2.1.2.3 for Figure 2.3. 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 can always be present in multi-document summarization. Typically this does not happen in the single-document case. Redundancy has a significant impact on the coherence and the cohesion of this new type of extract [Torr 14] (Page 109). Multi-document summarization is the extension of single document summarization, but like already said, redundancy is not the only issue. The primary pipeline is shown in Figure 2.15.

Multi-document input is 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

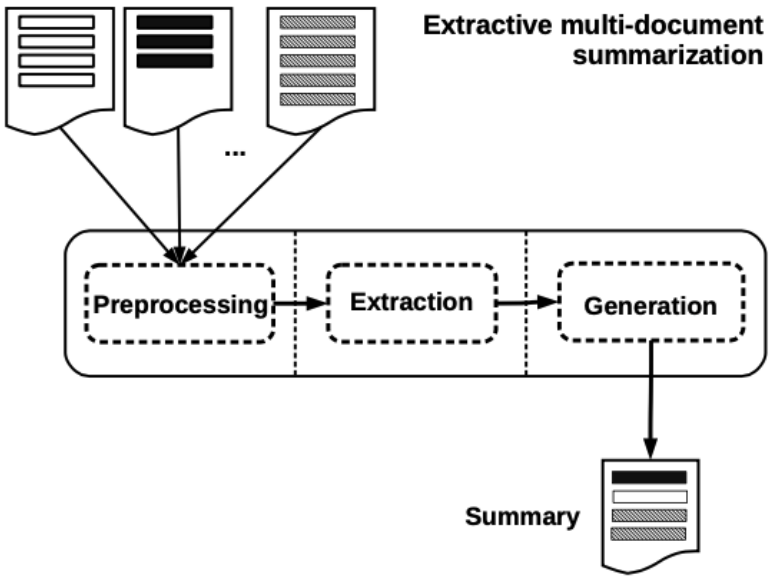


Figure 2.15.: Extraction based multidocument summarization system [Torr 14] (Page 110)

[Rade 98] (Pages 469-500). My case study example Google News is a typical example for multi-document summarization, because Google uses multiple sources of information (scrape news from different other websites), collect the information and process them through the pipeline (Figure 2.15). The key task is 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 = single document
n = the total number of single-documents

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
D = a single document
S = single sentence in a document D
m = number of sentences S in document D

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

$$T = t_1, t_2, ..., t_k$$

Where
t = a single token in the sentence S
T = all of the single tokens t

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 an informative sentence 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 2.15. 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 two steps. At first, as in 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 vector further feeds a subsequent algorithm that imputes the most related terms in the same topic-theme [Das 07] (Page 13). Once these 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 that 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 in 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 particular words known as the *topic signatures* [Luhn 58]. The log-likelihood is often used in statistical approaches for text summarization. There are two ways to calculate the importance of a sentence. The first way is as a function that contains the number of its topic signatures. Secondly, it can be calculated as the proportion of all the topic signatures in each sentence, respectively. While the first method usually gives higher scores to longer sentences, the second approach measures the occurrences of the topic words [Dunn 93].

Graph Spreading Activation is a similar approach to 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 research, is an informative summary. On the other hand, an indicative summary tries to transmit the relevant contents of the original document in such a way so that the readers can choose documents that match 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 crucial contents are 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, user-oriented systems produce personalized summaries that concentrate on specific information from the original documents only. For example, the user-oriented 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 any modification needed. On the 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

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 new 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 2.16 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 be a value like 15%, which means that the length of the original input document will be reduced by 85%.

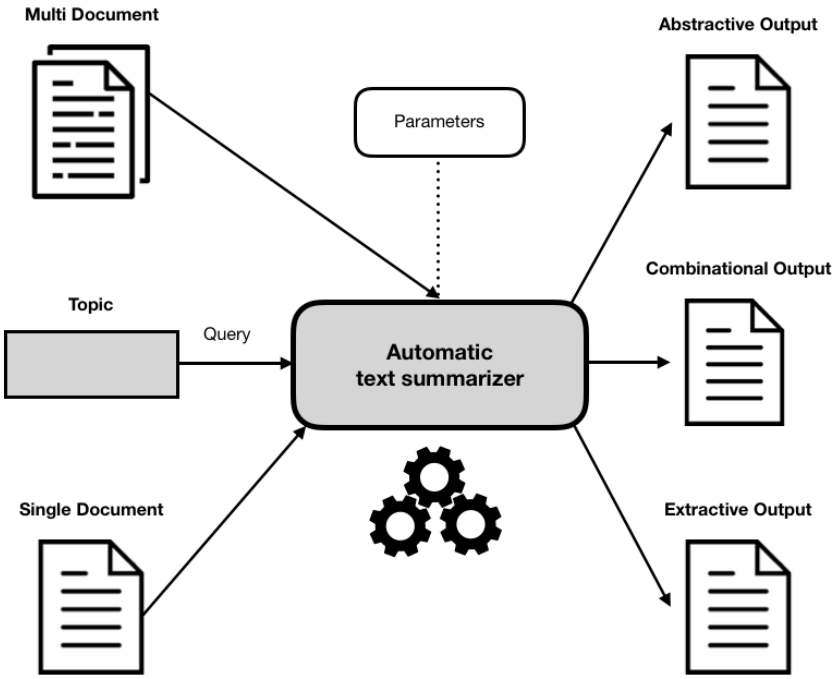


Figure 2.16.: Simplified Abstraction Extraction process

2.3.3.1. Extractive

The extractive approach aims to give an overview of the source document. This is done by selection fragments of the text (words, sentences, or paragraphs) that contain the essential information of the input text or texts. It can be seen as a copy and paste action for the most important fragments. Figure 2.17 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 original words or connectors. Furthermore, it cannot paraphrase as 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

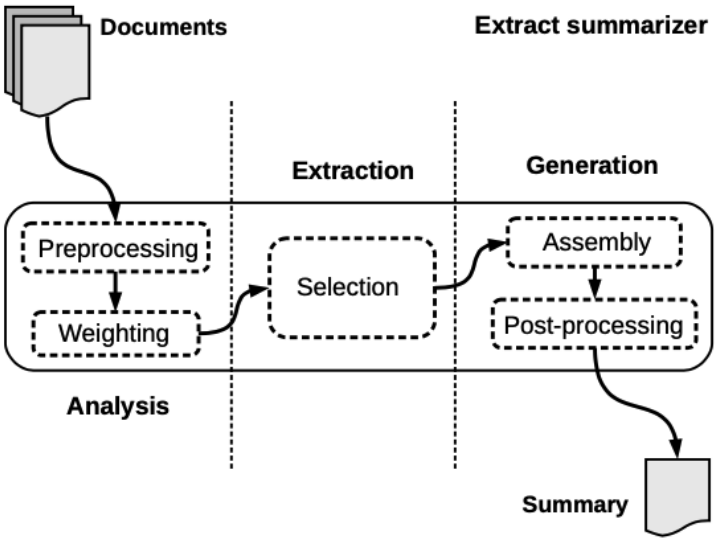


Figure 2.17.: General architecture of an extraction-based summarization system [Torr 14] (Page 31)

- Deep parsing techniques

The **surface-level** algorithms scratch on the right linguistic parts of the text. It cannot detect deep connections and abstractive features. It instead uses certain linguistic elements to detect the essential segments of a document [Torr 14] (Page 32). The mentioned inventor of automatic text summarizer Luhn [Luhn 58] used surface-level techniques to weight the occurrences of words in sentences. This technique is useful for a headline or keyword summarization.

The **intermediate-level** categorization digs deeper into the meaning of specific 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 Marcu published one of the earliest methods 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 weighs and orders the elements accordingly. The highest weighted elements will be selected for the summary.

Even though I proposed in 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 the latest achievements in the research for useful automated summaries. When a system is based on an abstractive method, it means that it profoundly 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 which 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 were a complex architecture to understand documents written in natural language, and it had a hard-coded so-called knowledge structure [DeJo 82] to simulate human behavior. However, the module for understanding the input was still conducted by a rather superficial analysis of the text. Frump's algorithm still made many 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 the 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 computational power. There are even ways to collaborate with the human, as in the **Aided Summarization**:

- Combines automatic methods with human input.
- The algorithm suggests important information from the input document or documents, and the human decides 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 translation model for the abstractive summarization model [Bank 00] (Pages 318-325). I mentioned at the beginning of the thesis that sometimes innovative approaches for one discipline, for example, machine translation can benefit from other disciplines (text summarization). A machine translation model converts a source language into the 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 the 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 summarization in a standardized way. This will be explained in the next section. The encoder-decoder model still faces some problems when using it directly for text summarization. It needs to be modified to work properly:

- Set the focus on the important sentences and keywords.
- Handle the new, rare, or made-up words in the source document.
- Handle a long document.
- Make a 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 and keywords, like the encoder-decoder, since attention is encoder-decoder based too.
- Handle the novel, rare or made-up words in the source document as well.

- Use beam-search to generate summary
- Use the n-gram match term as the loss function

Beam-search is 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 these methods achieve better results than the extractive approaches, but there are still even more advanced approaches possible (Section 2.4) that are based on the abstractive models.

2.3.4. Evaluation

Evaluating automatically generated summaries has always been a challenging task. New methods for evaluation were specially created for the summarization discipline. It is divided basically into two different approaches 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 a 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. Hence it 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 are 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 evaluation approach originated from machine translation is **BLEU**, which means Bilingual Evaluation Understudy. BLEU can also be applied for evaluating text summaries. In general, BLEU is calculated on the n-gram co-occurrence between the generated summary and the ideal human-written summary. It measures 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**: its suitability can measure the quality of a summary 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 of 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.

2.4. Advanced Approaches for Text Summarization

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

The four white boxes in Figure 2.18 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 entirely 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.

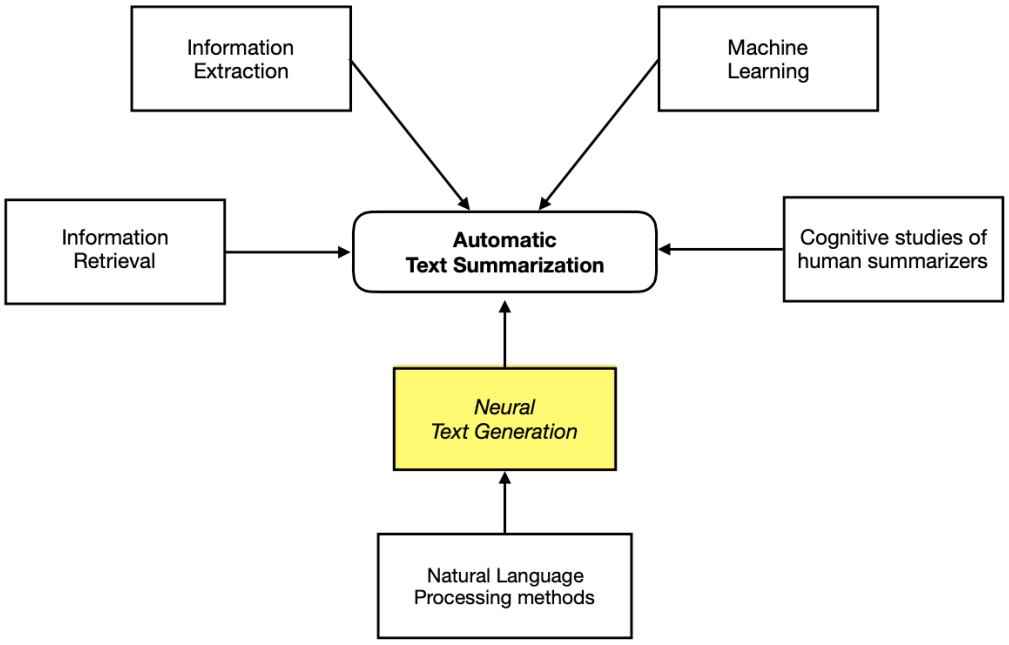


Figure 2.18.: Research fields with influence on the development of text summarization

2.4.1. Combinational Approach

- **Pointer-Generator Network**
- **Extract then Abstract model**

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

Figure 2.19 is based on the attention model (Figure 2.12 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 2.19). 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 the 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 extracts the meaningful sentences with an extractive approach and calculates then the relevance score for each word, according to the query uses that as input for a pre-trained abstractive model.

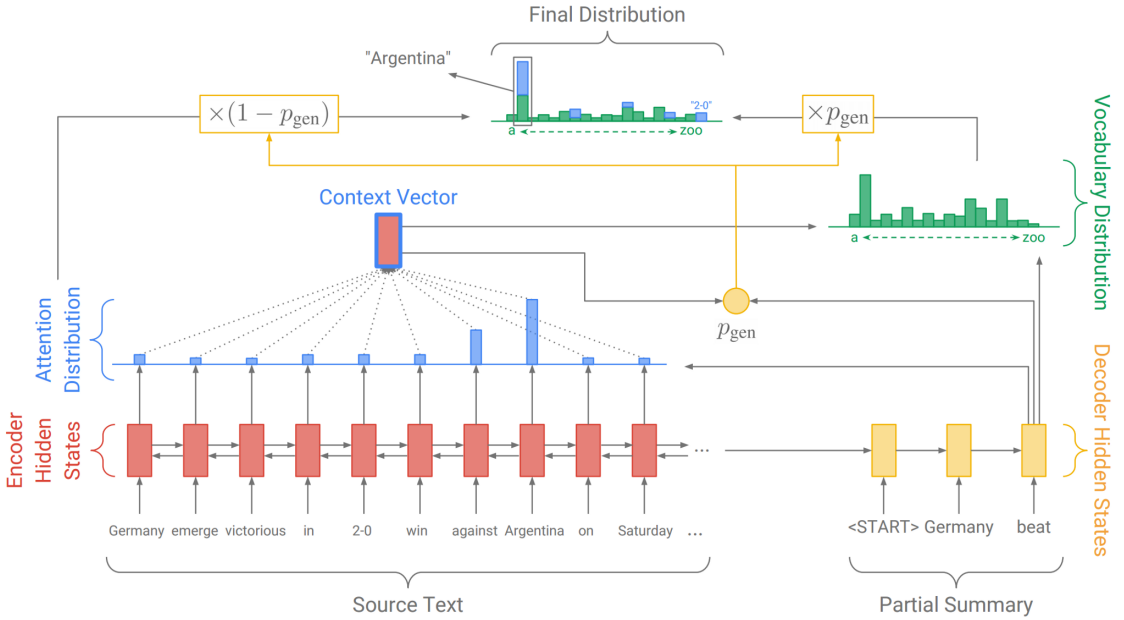


Figure 2.19.: The image describes the combination of the weights and the vocabulary distribution [See 17]

2.4.2. Transfer Learning

It is practical to reuse already learned models to make a new text summarization model. That is called *Transfer Learning*. It allows to build 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 a good representation of sentences. Several methods are proposed to create summaries by BERT.

Fine-tune BERT for Extractive Summarization

- Use the pretrained 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

Science is organized knowledge.
Wisdom is organized life.

Immanuel Kant

In this third section of my thesis, I finally present my self-programmed prototype. The State of the Art chapter was structured to focus on the necessary information for understanding my prototype, as shown in Figure 2.1. Hence I only gave some additional information about techniques that 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. There 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 flow 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 illustrate it further. 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 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 but related topic for my case study than my actual prototype to show another kind of task where automatic text summarization is useful.

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 giving 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.

This data-set contains around 500.000 entries with each entry containing one review from the amazon fine foods and its summary. The data includes a collection over eight years from October 2012 until now.

The training process of the algorithm must be finished in a reasonable amount of time. Due to the size of the data-set and the complexity of the neural network, it is usual to rent a cloud computer with a large graphics card. I regard this as an essential requirement that a strong GPU is available for the training process.

After finishing the training of the model, the algorithm must be capable of taking a random review-like text as an input and respond within a reasonable time (seconds) with the corresponding review summarization. Since I am not using the latest technologies, I don't always expect perfect grammatical correctness of the output. I regard the summary as a success if it summarizes the input without too much information loss. This measurement is subjective because the scoring metrics are not perfect for my case.

For example

Reasonable, Good summarization

Input: *I don't like this product at all! It looks completely different in reality than on the pictures. 0/5 Stars.*

Output: *Bad product*

Bad summarization

Input: *Not bad would describe this tea very well. It's more or less decent, but nothing special, I'm not sure if I can recommend it*

Output: *Very special tea*

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

3.2. Technical concept

An example review needs to be processed from the initially given data-set through many processes to be able to predict summaries. The technical concept illustrates all necessary steps without yet going into the programming detail itself. The purpose is to take all 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 preprocessing
- Building the Model
- Training the Model
- Generating Summary

These steps will be explained in the following Subsection.

3.2.1. Data Preprocessing

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 preprocessing 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 must be created. This dictionary can be used to convert the words into numbers, and respectively numbers are what the recurrent neural network wants as an input. For creating the two vocabulary dictionaries, it needs to be careful 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.

Consequently, 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 the algorithm can only this way detect if the word *not* is negatively connotated or not. If a review says something is not bad, then the algorithm can summarize it by computing the word good.

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 those words most likely do not play a vital role in predicting the output. Words like it, in or at not necessarily appear in the stopwords dictionary. For this reason, I remove them manually because they almost do not affect the meaning of the text, but after removing, they reduced the training time further.

After applying all of the preprocessing steps, the example will look like the following:

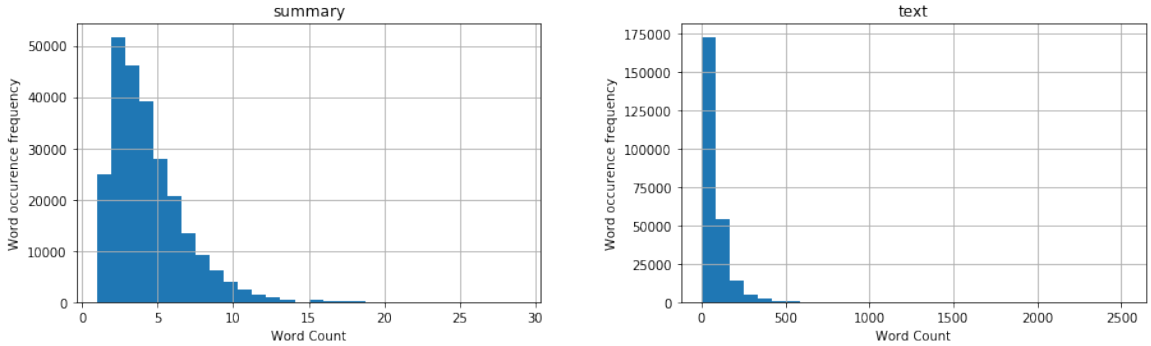


Figure 3.1.: Distribution of the sequences to estimate the maximum length for the review and summary

Input 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.

Cleaned Output 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 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. Because a fixed length is mandatory for building the model, this step is mandatory. Their dictionary number representation now replaces all of the words. If a review is shorter as the maximum length, the missing data points are filled with zeros.

Generated Word Vocabulary

['remember': 678, 'buying': 142, 'candy': 360, 'kid': 1017, 'quality': 87, 'dropped': 3724, 'years': 108, 'still': 55, 'superb': 1893, 'product': 9, 'disappointed': 243, 'delicious': 1, ...]

The dictionary contains the key-value pairs for all of the unique words in the trainable dataset. Those words are mapped into a tokenized and fixed length of 80 words padded sequence as shown below

All of the input sequences are padded into the same length. Too short sentences will be filled up with zeros as below, and too long sentences will be cut at exceeding the limit. The predicted summary has as well a fixed length.

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

In the same dictionary as above with a fixed length of 10

array([[1, 9, 0, 0, 0, 0, 0, 0, 0, 0]], dtype=int32)

The tokenized and padded review array can now be funneled into the neural network. This procedure needs to be done for all of the input sequences for the training, as well as the prediction process. This process has yet nothing to do with the defined methods from Figure 2.1.

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). For the purpose of building the model, I used the Deep Learning framework Keras. This framework allows the easy implementation of different Deep Learning cells like the RNN or the LSTM and even the Attention layer.

The mathematical notations for Figure 3.2 are shown in the box below. It shows one of the multiple internal steps inside the neural network. Looking at Figure 3.3, the **Encoder** of Figure 3.2 could be the computation starting from the *embedding* layer until the *input_2*.

The mathematical notations for Figure 3.2:

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

where $n = 80$
$\boldsymbol{x_1} = 678$, $\boldsymbol{x_2} = 142$, ... ,$\boldsymbol{x_{10}} = 243$, $\boldsymbol{x_{n-1}} = 0$

678 = 'remember' , ...

$[\boldsymbol{y_1}, \boldsymbol{y_2}, \boldsymbol{y_3}, ..., \boldsymbol{y_{Ty}}]$ where each x represents one word of the input sequence,

where $n = 10$
$\boldsymbol{y_1} = 1$, $\boldsymbol{y_2} = 9$, ... ,$\boldsymbol{y_{n-1}} = 0$

1 = 'delicious', 9 = 'product'

$\boldsymbol{T_x} = n$ from x, represents the fixed length of the input sequence

$\boldsymbol{T_y} = n$ from y, represents the fixed length of the output sequence

The reason for choosing the LSTM over an underlying recurrent neural network architecture is that the generated summary can sometimes be quite long (up to 80 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 an input or output has fewer words than the maximum length, the missing words are denoted as zero values. This enables the algorithm not to compute any further predictions because it is a zero multiplication. The encoder-decoder architecture is mainly used when the input and output length vary from each other.

This processing step is an independent process of the model building step. The output for this step is generated and ready to train the model. The model is built based on the structure from Figure 2.1. In order to use the attention layer for the neural network, it is necessary

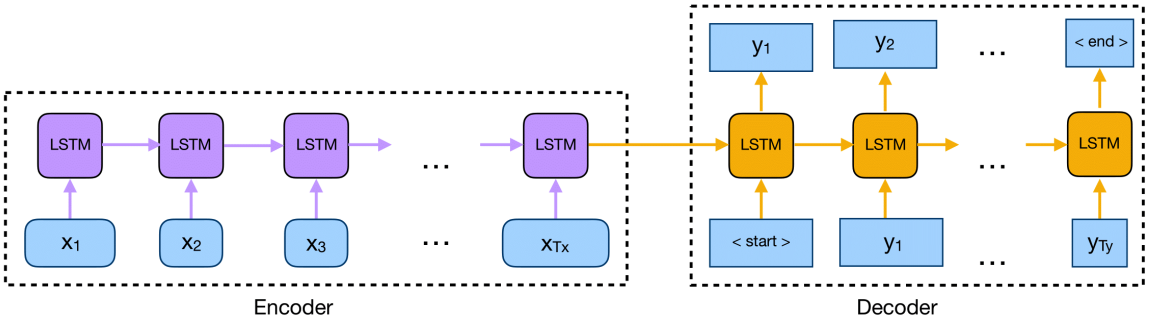


Figure 3.2.: Many to many Sequence to Sequence LSTM model

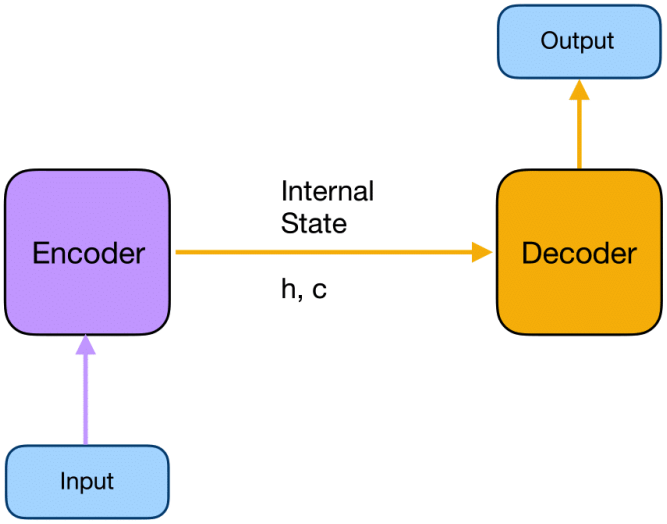


Figure 3.3.: Perspective of the text summarization

to use an underlying time-series capable layer like the LSTM cell, as discussed in Chapter 2. This is based on the Recurrent Neural Network, but it is not necessary to implement an RNN cell because the LSTM is just an advanced version of the RNN and substitutes it.

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

The mathematical notations for Figure 3.3:

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

The output summarized model looks now like Figure **??** with its 81.380.367 parameters. Where the input layer is responsible for defining the shape and dimension of the input sequence. The shape [(None, 80)] says that every trainable input must be in the shape of an array with a length of 80. The embedding layer transforms the real word (*remember buying ..*), respectively into its number-padded representation. The following layers are the LSTM layers, which the neural network the required complexity. Finally, the attention layer gives the additional *intelligence* to further increase the accuracy of the model. It can be considered as the *brain* of the summarization model.

Going back to the tabular overview in Table 2.1:

Text Generation Concepts

Architectures and Approaches:
Data-driven due to a large amount of data I have 500000, but only 224814 are used

Advanced Approaches for Text Generation

Attention:
Everything required for using the Attention layer is used in my prototype

Text Summarization Concepts

Input:
Multi-document due to the 224814 different training samples I am going to use. 500000 would take too much time.

Purpose:
Headline Summary due to the fixed-length output vector of length 10.

Output:
Abstractive approach due to the implemented attention layer, which gives the neural network the ability to chose from a wide range of words that are not used in the currently inputted review.

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 80)]              0
_________________________________________________________________
embedding (Embedding)        (None, 80, 500)           40049500
_________________________________________________________________
lstm (LSTM)                  [(None, 80, 500), (N 2002000
_________________________________________________________________
input_2 (InputLayer)         [(None, None)]            0
_________________________________________________________________
lstm_1 (LSTM)                [(None, 80, 500), (N 2002000
_________________________________________________________________
embedding_1 (Embedding)      (None, None, 500)         10933500
_________________________________________________________________
lstm_2 (LSTM)                [(None, 80, 500), (N 2002000
_________________________________________________________________
lstm_3 (LSTM)                [(None, None, 500),  2002000
lstm_2[0][1]
lstm_2[0][2]
_________________________________________________________________
attention_layer (AttentionLayer ((None, None, 500),  500500
lstm_3[0][0]
_________________________________________________________________
concat_layer (Concatenate)   (None, None, 1000)   0
attention_layer[0][0]
_________________________________________________________________
time_distributed (TimeDistribut (None, None, 21867)  21888867
=================================================================
Total params: 81,380,367
Trainable params: 81,380,367
Non-trainable params: 0
```

This listing visualizes the models summary for my Prototype with all corresponding parameters. Everything I explained in the model building section is summarized together in this 31 lines.

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). For achieving this, the prediction is a single 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

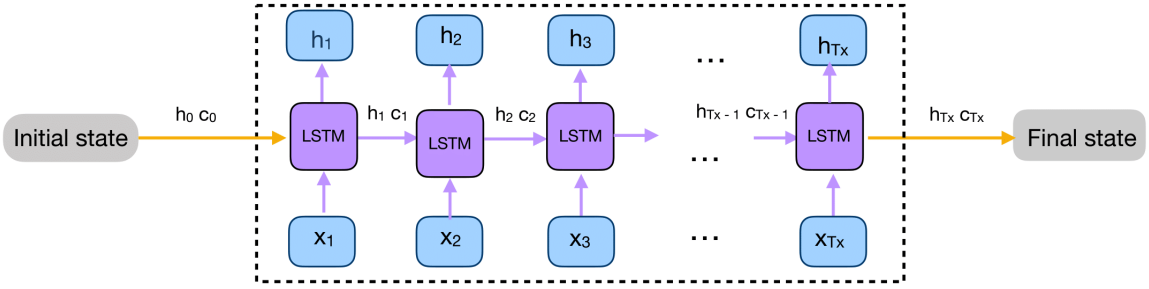


Figure 3.4.: Encoder of the LSTM

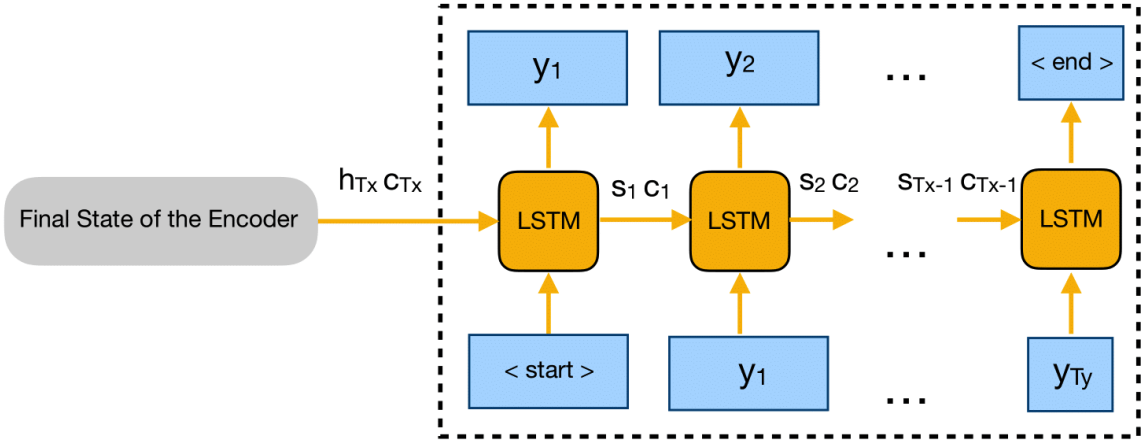


Figure 3.5.: Decoder of the LSTM

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 an LSTM network as well, which has the target sequence as a word-by-word input and predicts the same sequence offset by a single time step. The decoder 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. The target sequence is predicted by passing the first word into the decoder, which is always the *start* token. Moreover, the *end* token signals the end of the sentence, followed by potentially zero values.

Model training **Input**:

- An array of the size [224814, 80] representing 224814 unique training reviews encoded into the length 80.
- An array of the size [224814, 1] representing 224814 unique training review labels (ground truth) of the reviews. Their labels are the original summarizations for the given review. So the model can train on a review and its correct corresponding summarization in order to learn from that

Model training **Output**:

At this step, there is no concrete output, just a trained model with a huge size. The model can be saved in a special format, for example .h5. The size of this saved model is around 651,1MByte. This model can now be used to summarize my own reviews.

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, an 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
- 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.

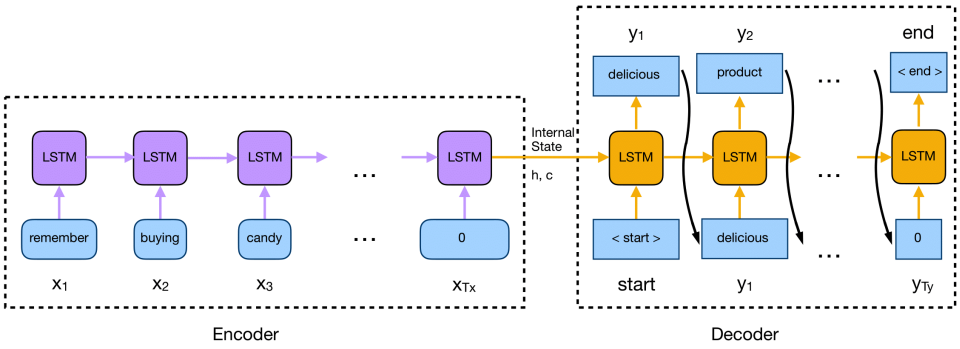


Figure 3.6.: Inference architecture to decode a sequence

Even an LSTM cell has limitations when it comes to sentence length. Since summaries can be quite long (my maximum length is 80 words), there needs to be made further improvements for the algorithm in order to work properly. It is not very easy 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 at all the words from the encoder output and decide which parts of the sentence the decoder wants to focus 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

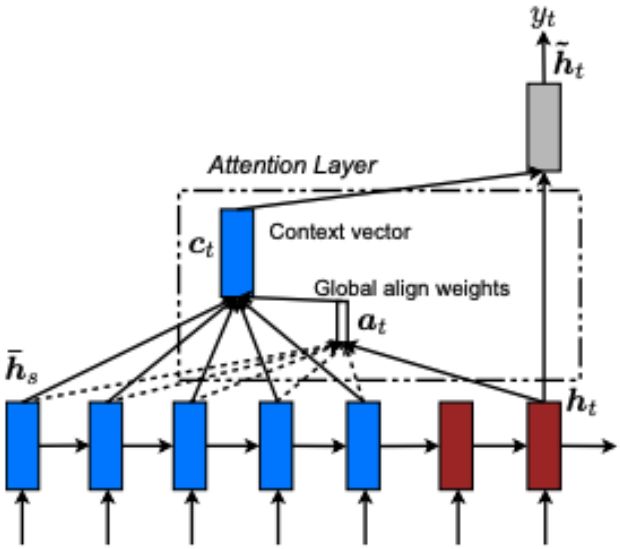


Figure 3.7.: Global Attention Model from [Minh 15]

Generate the new summary **Input**:

The inputted review must fulfill the same conditions as the training data. It needs to be encoded the *same way* as the training data was. So the same words must be transformed into their corresponding numerical representation. The length of 80 is mandatory too

Generate the new summary **Output**:

The output will be in the form of the padded y array of length 10. So a possible output could look like this:

$[23, 13, 5, 1, 0, 0, 0, 0, 0, 0]$

Those numbers must be back-transformed with the original dictionary into their corresponding words. Then the output is readable for humans.

3.3. Implementation

This is where the technical concepts comes to life. 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 an open-source platform. The python packages I used for this project are listed in the following Listing:

```
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 an 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 as mentioned 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

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

the following is the encoder part of the trainable model:

3.3.1. Data Preprocessing

The first step *data preprocessing* consists of multiple transformations of the training data. An example is:

```
stop_words = stopwords.words('english')
tokenizer = RegexpTokenizer(r'\w+')

def text_cleaner(text):
newString = text.lower()
#newString = BeautifulSoup(newString, "lxml").text
tags = re.compile('<.*?>|&([a-z0-9]+|#[0-9]{1,6}|#x[0-9a-f]{1,6});')
newString = tags.sub('', newString)
newString = re.sub(r'\([^)]*\)', '', newString)
newString = re.sub('"','', newString)
newString = ' '.join([contraction_mapping[t] if t in contraction_mapping
                                    else t for t in newString.split(" ")])
newString = re.sub(r"'s\b","",newString)
newString = re.sub("[^a-zA-Z]", " ", newString)
tokens = [w for w in newString.split() if not w in stop_words]
long_words=[]

for i in tokens:
  if len(i)>=3:                  #removing short word
    long_words.append(i)

return (" ".join(long_words)).strip()
```

Where this code refers to the example of Section 3.2.1. I can't explain every single step from the code above, but it performs exactly the transformation steps I have already explained. The last line of code returns a single preprocessed review as an entire sentence. The *.join()* and the *.strip()* commands assure that a single formatted sentence is returned as a string.

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.

Original provided Summary
Delicious product!

This python function will apply this text transformation. The cleaning transformation is applied both for the training reviews, as well as the corresponding available training summary. The next step is the tokenization from the same Section in the box **Generated Word Vocabulary**. This tokenized and padded review is created by funneling the cleaned summaries and reviews into the provided tokenization function from Keras.

```
#padding zero up to the maximum length
X_train = pad_sequences(X_train,  maxlen=max_len_text, padding='post')
X_test = pad_sequences(X_test, maxlen=max_len_text, padding='post')

x_voc_size = len(x_tokenizer.word_index) +1
```

3.3.2. Building the model

Building the model is essentially constructed by combining two different parts, as shown in Figure 3.3. I need to create the **Encoder**, and the **Decoder** on their own and combine them at the end.

```
# Encoder
# where max_len = 80 and x_voc_size = the vocabulary of 224814 reviews
encoder_inputs = Input(shape=(max_len,))
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. This is the **Encoder** part of Figure 3.6. 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 number 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 supported by Tensorflow; hence it needs to be imported through a third-party library or imported 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:

```
# 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
```

```
22 model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
```

This is the right part of Figure 3.6. In this way we completed the architecture of the text summarization model. The only remaining building step is to combine the encoder and the decoder. Finally this two parts are concatenated together as following:

```
1  model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
```

This creates the final model summary from the last Listing in Section 3.2.2.

3.3.3. Training the model

The training is now the easiest, but the most time-consuming part. The training process itself requires only a couple lines of code which specifies how to training should be proceeded:

```
1  history = model.fit([X_train, y_train[:,:-1]],
2  y_train.reshape(y_train.shape[0], y_train.shape[1], 1)[:,1:],
3      epochs=30,
4      callbacks=[es],
5      batch_size=512,
6      validation_data=([X_test,y_test[:,:-1]],
7          y_test.reshape(y_test.shape[0],y_test.shape[1], 1)[:,1:])
8  )
9  model.save('model.h5')
```

- *model.fit* calls the training process to start
- *epochs* sets the amount of so-called training rounds. The more the longer it takes
- *callbacks* provides a batch report during training of the process to stop early if necessary
- *batch_size* let 512 reviews train at the same time continuously
- *validation_data* provides a batch report during training of the model accuracy

The entire training process can be seen in Figure 3.8. The picture shows that my model finished due to the early stopping (callback function) at epoch 12. Each epoch took around 979 seconds, which is around 16 minutes. Multiplying this by 12 equals 3.5 hours of training. On a computer with a weaker graphics card, this training process can easily take five times as long. It highly depends on the available graphics power.

```
Train Model...
Train on 224814 samples, validate on 24980 samples
Epoch 1/30
224814/224814 [==============================] - 982s 4ms/sample - loss: 2.8669 - val_loss: 2.4505
Epoch 2/30
224814/224814 [==============================] - 980s 4ms/sample - loss: 2.3291 - val_loss: 2.2047
Epoch 3/30
224814/224814 [==============================] - 979s 4ms/sample - loss: 2.1111 - val_loss: 2.0710
Epoch 4/30
224814/224814 [==============================] - 979s 4ms/sample - loss: 1.9626 - val_loss: 1.9864
Epoch 5/30
224814/224814 [==============================] - 977s 4ms/sample - loss: 1.8385 - val_loss: 1.9306
Epoch 6/30
224814/224814 [==============================] - 976s 4ms/sample - loss: 1.7271 - val_loss: 1.8944
Epoch 7/30
224814/224814 [==============================] - 978s 4ms/sample - loss: 1.6239 - val_loss: 1.8619
Epoch 8/30
224814/224814 [==============================] - 979s 4ms/sample - loss: 1.5275 - val_loss: 1.8335
Epoch 9/30
224814/224814 [==============================] - 977s 4ms/sample - loss: 1.4366 - val_loss: 1.8225
Epoch 10/30
224814/224814 [==============================] - 975s 4ms/sample - loss: 1.3509 - val_loss: 1.8176
Epoch 11/30
224814/224814 [==============================] - 977s 4ms/sample - loss: 1.2701 - val_loss: 1.8145
Epoch 12/30
224814/224814 [==============================] - 976s 4ms/sample - loss: 1.1944 - val_loss: 1.8190
Epoch 00012: early stopping
Model saved
```

Figure 3.8.: My prototypes training process

3.3.4. Generating Summary

I compare my generated summary with the origin ground truth summary in the next Section 3.4, because the quality of the generation can either be measured by our human understanding or by other evaluation metrics mentioned in Section 3.4. The generated summaries are shown in Table 3.1.

3.4. Evaluation

The model and its output can be evaluated in 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 prevents a lot of memory issue related problems. 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 ten 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 that can be mapped

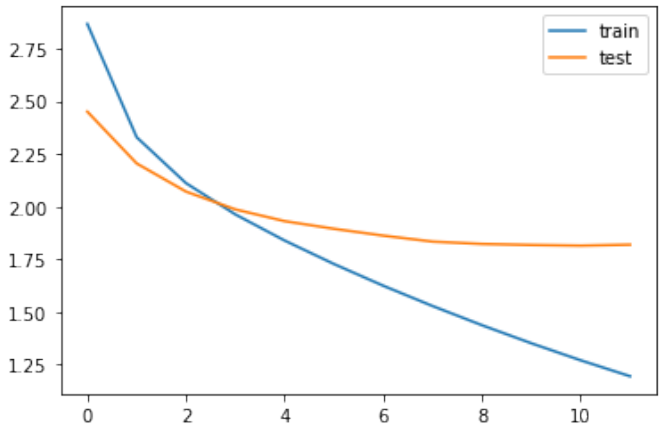


Figure 3.9.: Training and validation loss during the training phase

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 the quality of the summaries:

Predictionsq

The following predictions have been made with a training of 224814 samples and were validated on 24980 samples for the validation (testing) loss. After epoch 12, the training stopped due to early stopping, because the loss increased again instead of further decreasing.

BLEU Score

As introduced in Section 2.3.4, there are two main approaches for evaluating the accuracy of the generated summary. The first way is using the BLEU score, which was initially published for evaluating machine translation generations. The following Table 3.2 evaluates the predicted summaries from Table 3.1 with multiple Rouge Scores.

It can be seen that the Rouge score produces semantically incorrect scores because it only looks at whether the words in the original and predicted summaries are similar.

For that reason, the prediction

```
Original: "yummy sweet sherry vinegar"
Prediction: "love these"
```

Produces a Rouge score of 0, even though there is a semantic similarity. Whereas the summary

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

Scoring Method	Rouge-1	Rouge-2	Rouge-L	Rouge-BE
Original: too salty and dry Predicted: too salty	*0.667*	0	*0.667*	0
Original: at the movies and home Predicted: love it	0	0	0	0
Original: must be good Predicted: my dogs love these	0	0	0	0
Original: yummy sweet sherry vinegar Predicted: love these	0	0	0	0
Original: wrong coffee received Predicted: coffee received	*0.8*	*0.66*	*0.8*	*0.66*

Table 3.2.: The generated Sentences evaluated with the Rouge Score

Scoring Method	1-gram	2-gram	3-gram	4-gram
Original: too salty and dry **Predicted: too salty**	*0.367879*	0.367879	*0*	0
Original: at the movies and home **Predicted: love it**	0	0	0	0
Original: must be good **Predicted: my dogs love these**	0	0	0	0
Original: yummy sweet sherry vinegar **Predicted: love these**	0	0	0	0
Original: wrong coffee received **Predicted: coffee received**	*0.606531*	*0.606531*	*0.8*	*0.66*

Table 3.3.: The generated sentences evaluated with the BLEU Score

Original: "wrong coffee received"
Prediction: "coffee received"

Scores really high, even though the summary is incorrect. The summary model did not capture the dependency of **wrong**. For this reason, I tried evaluating the five summaries with the *BLEU Score* mentioned in Section 2.3.4.

BLEU Score

In the following Table 3.3, I calculated up to the *4-gram* BLEU Scores for the predictions in Table 3.1.

Since the BLEU Score is evaluating the *n-grams* of the texts, it behaves similarly as the Rouge Score. It can even be said that Rouge and BLEU are complementing each other, due of the BLEU measures how much of the words in the generated summary appear in the original summary, whereas the Rouge measure how much of the words in the original summary appear in the generated summary.

Conducting evaluations on extractive text summarizers turns out to be way easier because the extractive approach picks its weighted meaningful sentences and appends it to the summary. So it uses only the vocabulary from the trained text. The abstractive summarizer showed to be more complex to evaluate correctly due to its flexible structure. A recent paper in 2019 conducted research on how to properly evaluate an abstractive text summarizer. The team of Kryscinski showed that the current evaluation protocol reflects human judgments only in a week way, while it also fails to evaluate critical features, for example, factual correctness (semantic correctness) of text summarization [Krys 19].

Chapter 4.

Generation of transferable knowledge

Science never solves a problem
without creating ten more

George Bernard Shaw

This last chapter conducts some final thoughts about my bachelor thesis.

Building the model was not an easy task. For the training, I needed to rent a cloud computing system from Amazon AWS [1], otherwise my computer would have needed too long. Without renting a high power graphics card, the training process would have taken around 30 hours on my personal computer. I paid around 12$ for 10 hours of GPU (graphics card) power. With this rented power, the training process only took around 3 hours.

The predictions from the last Section 3.4 showed that the generated summaries could be much more accurate. There are multiple ways to further enhance the quality of the generated summaries with the following approaches.

4.1. Improving the models performance

4.1.1. First option

Increasing the training data. The entire dataset consists of 568.454 training data rows. For my prototype I only limited my training to 250.000 rows. Even that I took only half of the available training data, the strong Amazon AWS GPU still took 3 hours to train the model. The computation time rises monotonically with the provided training data, so if I had used all of the available data, the entire process would have taken around 7 hours.

It is not guaranteed that the quality of predicted summaries increases by using more training data, but this is by far the easiest screw to set for trying to improve the quality.

[1] https://aws.amazon.com/de/

4.1.2. Second option

I already used the advanced version of the Recurrent Neural Network, the LSTM. But even the LSTM can be already outdated by way more advanced time-series cells. One possible example is the Bidirectional LSTM. Using bidirectional will use the inputs in two ways. One from the past to future and one from the future to past. The Bidirectional LSTM is capable of preserving information from both the future words and the past words. This cell can understand context much better than the usual LSTM. Still, this is very complicated to implement and this exceeded my time limitation for this thesis.

4.1.3. Third option

Using the Beam-Search strategy from Section 2.3.3.2 can lead to better results. It starts by generating the first word, and keeps the 10 next most suitable sequences of words (beams) around and generates on top of them. But the generated words sometimes lack diversity. It needs to be tested whether the generated summaries have a higher quality with or without Beam-Search, but it is still worth a try.

4.1.4. Fourth option

Taken from the Section 2.4.1 *Combinational Approach* of the advanced techniques.

Pointer Generator Networks from Section 2.4.1 can improve the model's performance dramatically. Those networks build further on top of the attention layer. If we consider Figure 2.1, the Pointer Generator Network would be right after attention. The same way Sequence to Sequence models are needed for applying the attention layer, the attention layer is needed for implementing the Pointer Generator.

This is by far the most complicated option of those four, but the most state-of-the-art and the most promising one.

Figure 4.1 shows a comparison between the attention architecture in Figure 2.12 and the Pointer Generator Network. It shows the the architecture get further expanded on top attention. It is a hybrid network that can choose to copy words from the source via pointing, while at the same time allowing to retain generating words from our fixed vocabulary.

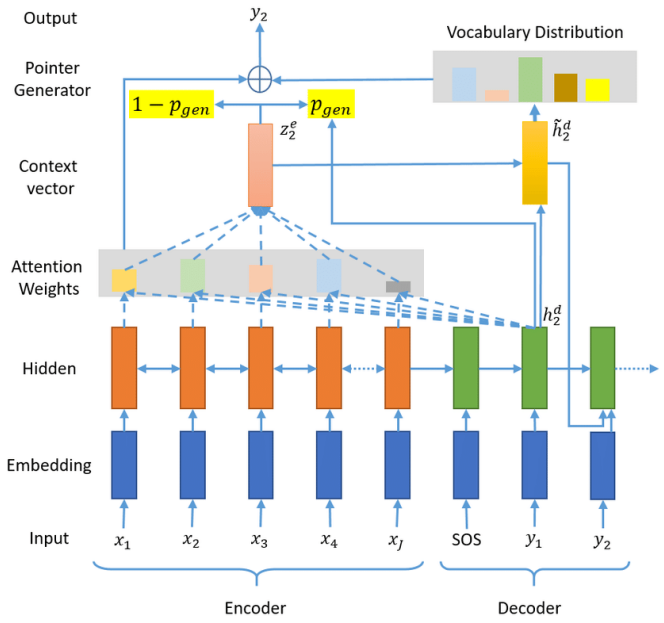


Figure 4.1.: Pointer Generator Network architecture from *https://www.researchgate.net/*

4.1.5. Fifth option

Taken from the Section 2.4.2 *Transfer Learning Approach* of the advanced techniques.

Using Googles BERT architecture is an entirely different approach. Since Google uses this approach to summarize its own news headlines, it obviously also works very well for my task. Still, this architecture works differently and the prototype needs to be structured in another way. For the sake of my prototype I stayed with the basic sequence to sequence, to better explain how this easy architecture works.

I would be interesting to see how BERT is doing in summarizing the food reviews.

4.2. Final Thoughts

Making changes like the improvement tips in the model architecture requires much knowledge, so the most comfortable but most expensive way is increasing the training data size first. It was a though task for me to fully understand what my prototype is doing in depth, for that reason I kept it as simple as possible. The other highly research topics are more applicable for a master thesis. Even that the summaries are beyond not perfect, I am still proud that they show a certain level of intelligence.

It made much fun to write this thesis and I learned so much new while investigating through text generation and text summarization. There is much more to explore and the Natural Language processing field is huge.

The name for my thesis was at first *IT-based Text Generation with the use of NLP Methods*. I noticed after the research that this title is not specific enough. Even in the range of Text Generation exist so many application fields, that the range I was searching in was just too wide. I decided to further decrease the scope of my thesis to be only about text summarization itself, but even that is probably not specific enough.

It is interesting to see how specific something becomes after digging deep into the research material.

This project can be cloned from Github under the following link:

https://github.com/Mavengence/Bachelor-Thesis-Textgeneration-TH.OHM

Thank you for reading through this thesis.

Written by

Tim Löhr

Appendix A.

Supplemental Information

I provided my Jupyter Notebook as a PDF, since I wrote my entire prototype in this Jupyter Notebook.

Bachelor_Thesis_Text_Generation

May 3, 2020

Bachelor Thesis

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

State of the art and design of a prototype

from Tim Löhr

1 1.0 Importing the Dependcies

[1]:
```
from attention import AttentionLayer

import numpy as np
import pandas as pd
import re
from bs4 import BeautifulSoup

from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.tokenize import RegexpTokenizer

from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
from keras import backend as K

from tensorflow.keras.layers import Input, LSTM, Embedding, Dense, Concatenate,␣
↪TimeDistributed, Bidirectional
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.models import load_model, model_from_json

from sklearn.model_selection import train_test_split

import nltk
```

```
nltk.download('stopwords')
import matplotlib.pyplot as plt

import spacy
spacy.load("en")

from sumeval.metrics.rouge import RougeCalculator
from sumeval.metrics.bleu import BLEUCalculator
from nltk.translate.bleu_score import sentence_bleu

import warnings
pd.set_option("display.max_colwidth", 200)
warnings.filterwarnings("ignore")
```

```
Using TensorFlow backend.
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/timloehr/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

2 2.0 Loading the Data

[4]:
```
data = pd.read_csv('amazon-fine-food-reviews/Reviews.csv', nrows=250000)
data.head(1)
```

[4]:
```
   Id   ProductId          UserId ProfileName  HelpfulnessNumerator  \
0   1  B001E4KFG0  A3SGXH7AUHU8GW  delmartian                     1

   HelpfulnessDenominator  Score        Time                Summary  \
0                       1      5  1303862400  Good Quality Dog Food

                                              Text
0  I have bought several of the Vitality canned dog food products and have found
them all to be of good quality. The product looks more like a stew than a
processed meat and it smells better. My Labr…
```

[4]:
```
data.shape
```

[4]:
```
(250000, 10)
```

3 3.0 Data Preprocessing

[5]:
```
data = data[['Summary', 'Text']]
```

[6]:
```
data.drop_duplicates(subset='Text')
data.dropna(axis=0, inplace=True)
```

[7]:
```
data.shape
```

[7]:
```
(249990, 2)
```

3.0.1 Contraction Mapping

[8]:
```
from contraction_mapping import contraction_mapping

contraction_mapping = contraction_mapping()
```

3.1 Cleaning

3.1.1 Text Cleaning

[9]:
```
stop_words = stopwords.words('english')
tokenizer = RegexpTokenizer(r'\w+')

def text_cleaner(text):
    newString = text.lower()
    #newString = BeautifulSoup(newString, "lxml").text
    tags = re.compile('<.*?>|&([a-z0-9]+|#[0-9]{1,6}|#x[0-9a-f]{1,6});')
    newString = tags.sub('', newString)
    newString = re.sub(r'\([^)]*\)', '', newString)
    newString = re.sub('"',''', newString)
    newString = ' '.join([contraction_mapping[t] if t in contraction_mapping␣
 ↪else t for t in newString.split(" ")])
    newString = re.sub(r"'s\b","",newString)
    newString = re.sub("[^a-zA-Z]", " ", newString)
    tokens = [w for w in newString.split() if not w in stop_words]
    long_words=[]
    for i in tokens:
        if len(i)>=3:                  #removing short word
            long_words.append(i)
    return (" ".join(long_words)).strip()
```

Cleaned text concatenate with DataFrame

[10]:
```
cleaned_text = []
for t in data['Text']:
    cleaned_text.append(text_cleaner(t))
```

3.1.2 Summary Cleaning

[11]:
```
data['Summary'][:10]
```

[11]:
```
0                          Good Quality Dog Food
1                              Not as Advertised
```

```
2                            "Delight" says it all
3                                   Cough Medicine
4                                      Great taffy
5                                       Nice Taffy
6    Great!  Just as good as the expensive brands!
7                           Wonderful, tasty taffy
8                                       Yay Barley
9                                 Healthy Dog Food
Name: Summary, dtype: object
```

[12]:
```
def summary_cleaner(text):
    newString = re.sub('"','', text)
    newString = ' '.join([contraction_mapping[t] if t in contraction_mapping␣
 ↪else t for t in newString.split(" ")])
    newString = re.sub(r"'s\b","",newString)
    newString = re.sub("[^a-zA-Z]", " ", newString)
    newString = newString.lower()
    tokens=newString.split()
    newString=''
    for i in tokens:
        if len(i)>1:
            newString=newString+i+' '
    return newString
```

[13]:
```
cleaned_summary = []
for t in data['Summary']:
    cleaned_summary.append(summary_cleaner(t))

data['cleaned_text']=cleaned_text
data['cleaned_summary']=cleaned_summary

data['cleaned_summary'].replace('', np.nan, inplace=True)
data.dropna(axis=0,inplace=True)

data['cleaned_summary'] = data['cleaned_summary'].apply(lambda x: '_START_ ' +␣
 ↪x + ' _END_')
```

[14]:
```
data.head()
```

[14]:
```
                  Summary  \
0  Good Quality Dog Food
1      Not as Advertised
2  "Delight" says it all
3         Cough Medicine
4            Great taffy

                                                Text  \
```

```
0  I have bought several of the Vitality canned dog food products and have found
them all to be of good quality. The product looks more like a stew than a
processed meat and it smells better. My Labr…
1           Product arrived labeled as Jumbo Salted Peanuts…the peanuts were
actually small sized unsalted. Not sure if this was an error or if the vendor
intended to represent the product as "Jumbo".
2  This is a confection that has been around a few centuries.  It is a light,
pillowy citrus gelatin with nuts - in this case Filberts. And it is cut into
tiny squares and then liberally coated with …
3  If you are looking for the secret ingredient in Robitussin I believe I have
found it.  I got this in addition to the Root Beer Extract I ordered (which was
good) and made some cherry soda.  The fl…
4                                                                  Great taffy at a
great price.  There was a wide assortment of yummy taffy.  Delivery was very
quick.  If your a taffy lover, this is a deal.

                                cleaned_text  \
0                                     bought several vitality canned dog food
products found good quality product looks like stew processed meat smells better
labrador finicky appreciates product better
1                                                                        product
arrived labeled jumbo salted peanuts peanuts actually small sized unsalted sure
error vendor intended represent product jumbo
2  confection around centuries light pillowy citrus gelatin nuts case filberts
cut tiny squares liberally coated powdered sugar tiny mouthful heaven chewy
flavorful highly recommend yummy treat famil…
3
looking secret ingredient robitussin believe found got addition root beer
extract ordered made cherry soda flavor medicinal
4
great taffy great price wide assortment yummy taffy delivery quick taffy lover
deal

                        cleaned_summary
0  _START_ good quality dog food  _END_
1      _START_ not as advertised  _END_
2    _START_ delight says it all  _END_
3         _START_ cough medicine  _END_
4            _START_ great taffy  _END_
```

3.1.3 Distribution of the sequences

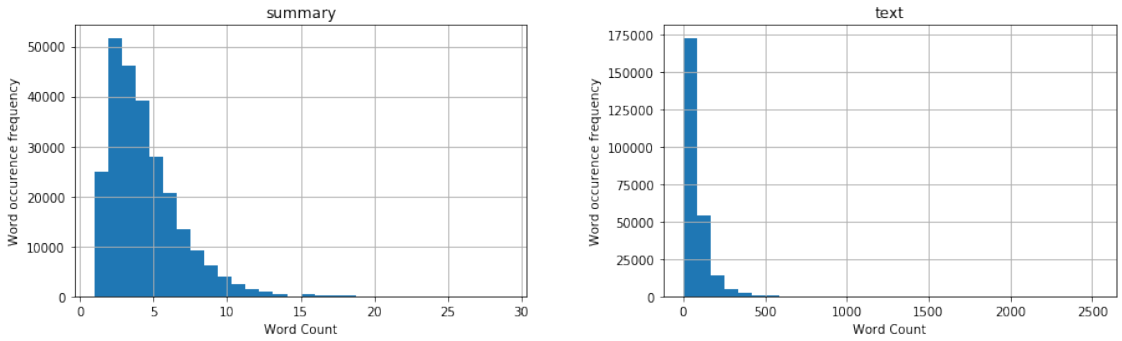
[22]:
```
text_word_count = []
summary_word_count = []

# populate the lists with sentence lengths
for i in data['Text']:
```

```
      text_word_count.append(len(i.split()))

for i in data['Summary']:
      summary_word_count.append(len(i.split()))

length_df = pd.DataFrame({'text':text_word_count, 'summary':summary_word_count})
axarr = length_df.hist(bins = 30, figsize=(15,4))
for ax in axarr.flatten():
    ax.set_xlabel("Word Count")
    ax.set_ylabel("Word occurence frequency")
plt.show()
```



[101]:
```
max_len_text=80
max_len_summary=10
```

3.1.4 Preparing Tokenizer

[102]:
```
X_train , X_test , y_train , y_test = train_test_split(data['cleaned_text'],␣
 ↪data['cleaned_summary'], test_size=0.1, random_state=0, shuffle=True)

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(224814,)
(24980,)
(224814,)
(24980,)
```

Text Tokenizer

[103]:
```
#prepare a tokenizer for reviews on training data
x_tokenizer = Tokenizer()
x_tokenizer.fit_on_texts(list(X_train))
```

```
#convert text sequences into integer sequences
X_train = x_tokenizer.texts_to_sequences(X_train)
X_test = x_tokenizer.texts_to_sequences(X_test)

#padding zero upto maximum length
X_train = pad_sequences(X_train,  maxlen=max_len_text, padding='post')
X_test = pad_sequences(X_test, maxlen=max_len_text, padding='post')

x_voc_size = len(x_tokenizer.word_index) +1
```

Summary Tokenizer

[104]:
```
#preparing a tokenizer for summary on training data
y_tokenizer = Tokenizer()
y_tokenizer.fit_on_texts(list(y_train))

#convert summary sequences into integer sequences
y_train = y_tokenizer.texts_to_sequences(y_train)
y_test = y_tokenizer.texts_to_sequences(y_test)

#padding zero upto maximum length
y_train = pad_sequences(y_train, maxlen=max_len_summary, padding='post')
y_test = pad_sequences(y_test, maxlen=max_len_summary, padding='post')

y_voc_size = len(y_tokenizer.word_index) +1
```

[105]:
```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(224814, 80)
(24980, 80)
(224814, 10)
(24980, 10)
```

4 4.0 Model

[106]:
```
K.clear_session()
latent_dim = 500

# 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_sequences=True, return_state=True)
encoder_outputs, state_h, state_c= encoder_lstm3(encoder_output2)

# Set up the 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)
```

[107]:
```
model.summary()
```

```
Model: "model"
__________________________________________________________________________________
__________________
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
Trainable params: 81,380,367
Non-trainable params: 0
__________________________________________________________________________________
__________________
```

Model optimization

[108]:
```
model.compile(optimizer='rmsprop', loss='sparse_categorical_crossentropy')

es = EarlyStopping(monitor='val_loss', mode='min', verbose=1)
```

Model fitting

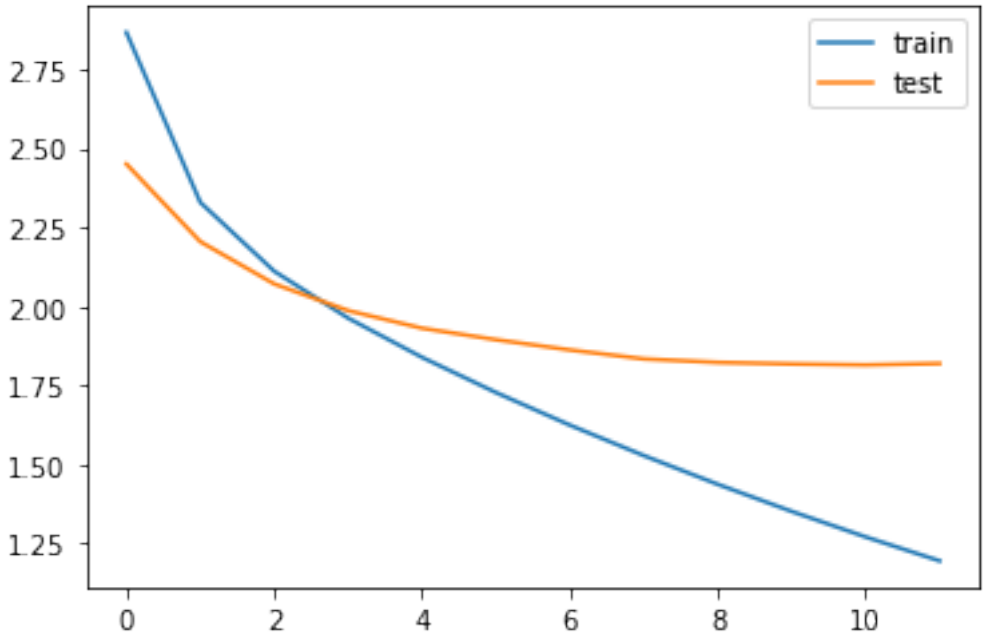
[109]:
```
try:
    model = load_model('model.h5', custom_objects={'AttentionLayer':␣
↪AttentionLayer})
    print("Model successfully loaded.")
except:
    print("Train Model...")
    history = model.fit([X_train, y_train[:,:-1]],
                        y_train.reshape(y_train.shape[0], y_train.shape[1], 1)[:,1:
↪],
                        epochs=30,
                        callbacks=[es],
                        batch_size=512,
                        validation_data=([X_test,y_test[:,:-1]],
                                         y_test.reshape(y_test.shape[0],y_test.
↪shape[1], 1)[:,1:])
                       )
    model.save('model.h5')
    print("Model saved")
```

```
Train Model…
Train on 224814 samples, validate on 24980 samples
Epoch 1/30
224814/224814 [==============================] - 982s 4ms/sample - loss: 2.8669
- val_loss: 2.4505
Epoch 2/30
224814/224814 [==============================] - 980s 4ms/sample - loss: 2.3291
- val_loss: 2.2047
Epoch 3/30
224814/224814 [==============================] - 979s 4ms/sample - loss: 2.1111
- val_loss: 2.0710
Epoch 4/30
224814/224814 [==============================] - 979s 4ms/sample - loss: 1.9626
- val_loss: 1.9864
Epoch 5/30
224814/224814 [==============================] - 977s 4ms/sample - loss: 1.8385
- val_loss: 1.9306
Epoch 6/30
224814/224814 [==============================] - 976s 4ms/sample - loss: 1.7271
- val_loss: 1.8944
Epoch 7/30
224814/224814 [==============================] - 978s 4ms/sample - loss: 1.6239
- val_loss: 1.8619
Epoch 8/30
224814/224814 [==============================] - 979s 4ms/sample - loss: 1.5275
- val_loss: 1.8335
Epoch 9/30
```

```
224814/224814 [==============================] - 977s 4ms/sample - loss: 1.4366
- val_loss: 1.8225
Epoch 10/30
224814/224814 [==============================] - 975s 4ms/sample - loss: 1.3509
- val_loss: 1.8176
Epoch 11/30
224814/224814 [==============================] - 977s 4ms/sample - loss: 1.2701
- val_loss: 1.8145
Epoch 12/30
224814/224814 [==============================] - 976s 4ms/sample - loss: 1.1944
- val_loss: 1.8190
Epoch 00012: early stopping
Model saved
```

5 5.0 Prediction

[110]:
```
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()
```



[111]:
```
reverse_target_word_index = y_tokenizer.index_word
reverse_source_word_index = x_tokenizer.index_word
target_word_index = y_tokenizer.word_index
```

Inference

[112]:
```
# encoder inference
encoder_model = Model(inputs=encoder_inputs, outputs=[encoder_outputs, state_h,␣
↪state_c])

# decoder inference
# Below tensors will hold the states of the previous time step
decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
decoder_hidden_state_input = Input(shape=(max_len_text,latent_dim))

# Get the embeddings of the decoder sequence
dec_emb2= dec_emb_layer(decoder_inputs)

# To predict the next word in the sequence, set the initial states to the␣
↪states from the previous time step
decoder_outputs2, state_h2, state_c2 = decoder_lstm(dec_emb2,␣
↪initial_state=[decoder_state_input_h, decoder_state_input_c])

#attention inference
attn_out_inf, attn_states_inf = attn_layer([decoder_hidden_state_input,␣
↪decoder_outputs2])
decoder_inf_concat = Concatenate(axis=-1, name='concat')([decoder_outputs2,␣
↪attn_out_inf])

# A dense softmax layer to generate prob dist. over the target vocabulary
decoder_outputs2 = decoder_dense(decoder_inf_concat)

# Final decoder model
decoder_model = Model(
    [decoder_inputs] + [decoder_hidden_state_input, decoder_state_input_h,␣
↪decoder_state_input_c],
    [decoder_outputs2] + [state_h2, state_c2]
)
```

Inference Process

[123]:
```
def decode_sequence(input_seq):
    # Encode the input as state vectors.
    e_out, e_h, e_c = encoder_model.predict(input_seq)

    # Generate empty target sequence of length 1.
    target_seq = np.zeros((1,1))

    # Chose the 'start' word as the first word of the target sequence
    target_seq[0, 0] = target_word_index['start']
```

```
    stop_condition = False
    decoded_sentence = ''

    while not stop_condition:

        output_tokens, h, c = decoder_model.predict([target_seq] + [e_out, e_h,␣
↪e_c])

        # Sample a token
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_token = reverse_target_word_index[sampled_token_index]

        if(sampled_token != 'end'):
            decoded_sentence += ' ' + sampled_token

            # Exit condition: either hit max length or find stop word.
        if (sampled_token == 'end' or len(decoded_sentence.split()) >=␣
↪(max_len_summary-1)):
            stop_condition = True

        # Update the target sequence (of length 1).
        target_seq = np.zeros((1,1))
        target_seq[0, 0] = sampled_token_index

        # Update internal states
        e_h, e_c = h, c

    return decoded_sentence
```

[124]:
```
def seq2summary(input_seq):
    newString = ''

    for i in input_seq:
        if((i!=0 and i!= target_word_index['start']) and i !=␣
↪target_word_index['end']):
            newString = newString + reverse_target_word_index[i] + ' '

    return newString

def seq2text(input_seq):
    newString = ''

    for i in input_seq:
        if(i != 0):
            newString = newString + reverse_source_word_index[i] + ' '

    return newString
```

[125]:
```
for i in range(10):
    print("Review:", seq2text(X_test[i]))
    print("Original summary:", seq2summary(y_test[i]))
    print("Predicted summary:", decode_sequence(X_test[i].
 ↪reshape(1,max_len_text)))
    print("\n")
```

```
Review: already big fan popchips salt vinegar flavor saw coming chili lime
flavor excited try flavor disappoint tangy spicy dash sweet new favorite chip
never tried popchips aware texture different regular potato chips somewhere
traditional chip rice cake throw die hard potato chip fans want something
crunchy awesome tasting without many calories fat going new favorite snack
Original summary: excellent
Predicted summary:  delicious


Review: ordered chips found salty dry huge amount spices ball one bags opened
Original summary: too salty and dry
Predicted summary:  too salty


Review: found tea favorite movie theater found perfect tea guests everyone loves
makes love
Original summary: at the movies and home
Predicted summary:  love it


Review: dogs special diet treats feed favorites cause problems
Original summary: must be good
Predicted summary:  my dogs love these


Review: active lab loves chew really enjoys treats ration gags time usually give
treat sitting back porch relaxing chewing tennis ball really seem improve breath
inevitably pushes far back mouth gags throws seen real difference teeth
reduction plaque tartar keep hoping
Original summary: makes breath smell better dog always gags on them
Predicted summary:  great for puppy teeth


Review: want chocolate bar probably buy one want chocolate chip cookie ice cream
buy suggest want chocolate chip cookies try another kind cookie sort half baked
cookie effect makes although tasting good something eat many fast healthy
desirable
Original summary: cookie coated chocolate bars
Predicted summary:  good but not great
```

```
Review: delicious sherry flavor salad dressing great used marinade give try
sweet balsamic tart red wine vinegar
Original summary: yummy sweet sherry vinegar
Predicted summary:  love these


Review: cats loved treats think really help releasing hairballs noticed changes
litterbox prove treats help hairballs issues going buy treats cats finish
Original summary: my cats love temptations
Predicted summary:  my cats love these treats


Review: received medium roast receive correct coffee shown picture disappointed
suppose ill try lot trouble return
Original summary: wrong coffee received
Predicted summary:  coffee received


Review: kids love happybaby tots tried every flavor eat love getting good
organic nutrition ingredients wholesome convenient throw diaper bag purse stick
lunch box snack use spoon sometimes bowl home self feeding little one also give
pouch eat directly squeeze pouch thank happybaby great products
Original summary: love all happybaby tots
Predicted summary:  love all happybaby tots
```

6 6.0 Evaluation

6.1 6.1 Rouge Score

[78]:
```
rouge = RougeCalculator(stopwords=True, lang="en")

original_summary = ["too salty and dry", "at the movies and home", "must be␣
 ↪good", "yummy sweet sherry vinegar", "wrong coffee received"]
predicted_summary = ["too salty", "love it", "my dogs love these", "love␣
 ↪these", "coffee received"]
```

[109]:
```
for orig, pred in zip(original_summary, predicted_summary):
    rouge_1 = rouge.rouge_n(summary=orig, references=pred,n=1)

    rouge_2 = rouge.rouge_n(summary=orig, references=pred, n=2)

    rouge_l = rouge.rouge_l(summary=orig, references=pred)

    rouge_be = rouge.rouge_be(summary=orig, references=pred)
```

```
    print(40*"=")
    print("Original: " + orig)
    print("Predicted: " + pred)
    print("ROUGE-1: {}, ROUGE-2: {}, ROUGE-L: {}, ROUGE-BE: {}".format(rouge_1,␣
↪rouge_2, rouge_l, rouge_be).replace(", ", "\n"))
    print(40*"=")
    print("\n")
```

```
========================================
Original: too salty and dry
Predicted: too salty
ROUGE-1: 0.6666666666666666
ROUGE-2: 0
ROUGE-L: 0.6666666666666666
ROUGE-BE: 0
========================================


========================================
Original: at the movies and home
Predicted: love it
ROUGE-1: 0
ROUGE-2: 0
ROUGE-L: 0
ROUGE-BE: 0
========================================


a.dogs=(nsubj)=>love
<BasicElement: dogs-[nsubj]->love>
========================================
Original: must be good
Predicted: my dogs love these
ROUGE-1: 0
ROUGE-2: 0
ROUGE-L: 0
ROUGE-BE: 0
========================================


========================================
Original: yummy sweet sherry vinegar
Predicted: love these
ROUGE-1: 0
ROUGE-2: 0
ROUGE-L: 0
```

```
ROUGE-BE: 0
========================================


b.wrong=(amod)=>coffee
a.coffee=(nsubj)=>received
<BasicElement: coffee-[nsubj]->receive>
a.coffee=(nsubj)=>received
<BasicElement: coffee-[nsubj]->receive>
========================================
Original: wrong coffee received
Predicted: coffee received
ROUGE-1: 0.8
ROUGE-2: 0.6666666666666666
ROUGE-L: 0.8
ROUGE-BE: 0.6666666666666666
========================================
```

6.2 6.2 BLEU Score

[110]:
```
for reference, candidate in zip(original_summary, predicted_summary):

    reference_s = [reference.split()]
    candidate_s = candidate.split()

    print(40*"=")
    print("Original: " + reference)
    print("Prediction: " + candidate)
    print('Individual 1-gram: %f' % sentence_bleu(reference_s, candidate_s,␣
↪weights=(1, 0, 0, 0)))
    print('Individual 2-gram: %f' % sentence_bleu(reference_s, candidate_s,␣
↪weights=(0, 1, 0, 0)))
    print('Individual 3-gram: %f' % sentence_bleu(reference_s, candidate_s,␣
↪weights=(0, 0, 1, 0)))
    print('Individual 4-gram: %f' % sentence_bleu(reference_s, candidate_s,␣
↪weights=(0, 0, 0, 1)))
    print(40*"=")
    print("\n")
```

```
========================================
Original: too salty and dry
Prediction: too salty
Individual 1-gram: 0.367879
Individual 2-gram: 0.367879
Individual 3-gram: 0.000000
Individual 4-gram: 0.000000
```

```
======================================


======================================
Original: at the movies and home
Prediction: love it
Individual 1-gram: 0.000000
Individual 2-gram: 0.000000
Individual 3-gram: 0.000000
Individual 4-gram: 0.000000
======================================


======================================
Original: must be good
Prediction: my dogs love these
Individual 1-gram: 0.000000
Individual 2-gram: 0.000000
Individual 3-gram: 0.000000
Individual 4-gram: 0.000000
======================================


======================================
Original: yummy sweet sherry vinegar
Prediction: love these
Individual 1-gram: 0.000000
Individual 2-gram: 0.000000
Individual 3-gram: 0.000000
Individual 4-gram: 0.000000
======================================


======================================
Original: wrong coffee received
Prediction: coffee received
Individual 1-gram: 0.606531
Individual 2-gram: 0.606531
Individual 3-gram: 0.000000
Individual 4-gram: 0.000000
======================================
```

List of Figures

List of Tables

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)”.

[Krys 19] W. Kryscinski, N. S. Keskar, B. McCann, C. Xiong, and R. Socher. “Neural Text Summarization: A Critical Evaluation”. 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. 540–551, Association for Computational Linguistics, Hong Kong, China, Nov. 2019.

[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.