Prague University of Economics and Business

Faculty of Informatics and Statistics

Abstractive summarization of fact check reports with pre-trained transformer tuning on extractive summaries

MASTER THESIS

Study program:	Applied Informatics
Field of study:	Knowledge and Web Technologies

Author: Bc. Peter Vajdečka

Supervisor: prof. Ing. Vojtěch Svátek, Dr.

Prague, May 2022

Declaration

I declare that I have prepared my thesis *Abstractive summarization of fact check reports with pre-trained transformer tuning on extractive summaries* independently using the sources and literature mentioned in the thesis.

In Prague on 2.5.2022 . . . .

Student’s signature

Acknowledgements

Sincere thanks to prof. Ing. Vojtěch Svátek, Dr. for the factual advice and professional guidance of this work. Furthermore, I thank my fiancée Erika and my family for their support and finally God for the enlightenment of my mind and the strength to finish the work on time.

Abstract

Fact checking is an activity aiming to remedy the global problem of disinformation spread. The result of this process, undertaken by numerous initiatives such as demagog.cz or politifact.com, are fact check reports written by human editors. Since the reports are frequently too long for a casual reader, and contain auxiliary parts not directly relevant for judging the claim veracity, automated creation of fact check report summaries is a topical task. The reader could then look at the shorter summary, containing the most salient points of the report, and then decide whether they dig deeper into some parts of the full report or not.

In the field of natural language processing, neural network models with transformer architectures achieve state-of-the-art results on many downstream tasks, including text summarization. These models are trained on a massive textual knowledge base, which ensures that just a small quantity of data is required to fine-tune these models – in contrast to large amounts of training data needed when the learning process starts from scratch just for the particular application.

We propose a novel procedure for text data reduction for the purpose of fine-tuning a natural language generation model, the UNIFIED TEXT TO TEXT TRANSFORMER (T5), in order to summarize a fact check report. First, the *Local Outlier Factor approach* is used to generate an extractive summary of the report, using sentence vectorization via the TF-IDF, DOC2VEC and BERT contextual representations. In addition, BERT is fine-tuned specifically for the given task and achieves the best results when compared to the other vector representations. Finally, the T5 Transformer is fine-tuned using these extractive summaries (reports containing fewer sentences than the original ones) to generate the final abstractive summaries. On English texts from politifact.com, the new method outperformed all state-of-the-art methods. As regards the Czech language, we were, to our knowledge, the first to apply automatic summarization to demagog.cz data. For comparison, the new procedure was also applied to generate short summaries for a known Czech news dataset (SumeCzech); although we only used 10 % of the initial training data for model fine-tuning, we overcame most of the state-of-the-art results.

Keywords

natural language processing, natural language generation, neural network, local outlier factor, transformer architecture, BERT, TF-IDF, DOC2VEC, fact-checking, summarization

Abstrakt

Overovanie faktov je činnosť zameraná na riešenie globálneho problému šírenia dezinformácií. Výsledkom tohto procesu, ktorý realizujú mnohé iniciatívy, ako napríklad demagog.cz alebo politifact.com, sú správy o overovaní faktov, ktoré píšu ľudskí redaktori. Keďže správy sú často príliš dlhé pre bežného čitateľa a obsahujú pomocné časti, ktoré nie sú priamo relevantné pre posúdenie pravdivosti tvrdení, je aktuálnou úlohou automatizované vytváranie súhrnov správ o kontrole faktov. Vzhľadom na to, by si čitateľ potom mohol pozrieť kratšie zhrnutie, ktoré obsahuje najpodstatnejšie body správy a rozhodnúť sa, či sa bude hlbšie zaoberať niektorými časťami celej správy alebo nie.

V oblasti spracovania prirodzeného jazyka dosahujú modely neurónových sietí s transformer architektúrou špičkové výsledky pri mnohých úlohách vrátane sumarizácie textu. Tieto modely sa trénujú na obrovskej báze textových znalostí, čo zabezpečuje, že na vyladenie týchto modelov je potrebné len malé množstvo údajov, na rozdiel od veľkého množstva trénovaných údajov potrebných v prípade, keď sa proces učenia začína od nuly len pre konkrétnu aplikáciu.

Navrhujeme nový postup redukcie textových údajov na účely jemného doladenia modelu generovania prirodzeného jazyka - UNIFIED TEXT TO TEXT TRANSFORMER (T5) - s cieľom zhrnúť správu o kontrole faktov. Najprv sa na generovanie extraktívneho zhrnutia správy používa prístup *Lokálnej miery odľahlosti*, pričom sa využíva vektorizácia viet prostredníctvom kontextových reprezentácií TF-IDF, DOC2VEC a BERT. Okrem toho je BERT vyladený špeciálne pre danú úlohu a dosahuje najlepšie výsledky v porovnaní s ostatnými vektorovými reprezentáciami. Nakoniec sa pomocou týchto extrakčných súhrnov (správy obsahujúce menej viet ako pôvodné správy) doladí transformátor T5, aby sa vytvorili konečné abstraktné súhrny. Na anglických textoch zo stránky politifact.com prekonala nová metóda všetky najmodernejšie metódy. Pokiaľ ide o češtinu, podľa našich vedomostí sme ako prví aplikovali automatickú sumarizáciu na údaje z portálu demagog.cz. Pre porovnanie sme nový postup použili aj na generovanie krátkych súhrnov pre známy český súbor spravodajských dát (SumeCzech); hoci sme na doladenie modelu použili len 10 % počiatočných trénovaných dát, prekonali sme väčšinu špičkových výsledkov.

Klíčová slova

spracovanie prirodzeného jazyka, generovanie prirodzeného jazyka, neurónová sieť, lokálny odľahlý faktor, transformer architektúra, BERT, TF-IDF, DOC2VEC, kontrola faktov, sumarizácia

Contents

List of Figures

List of Tables

Introduction

There have been significant societal limits and other social restrictions in recent years as a result of the COVID-19 outbreak. To make matters worse, this has been exacerbated in recent months by Ukraine's military upheaval. In some areas of Ukraine, ordinary citizens are regrettably cut off from information. On the other hand, while the majority of the world's population does not face this issue, they are quite likely to be impacted by erroneous or difficult-to-verify statements. Unfortunately, this is bringing society not only into a military and economic crisis, but also into a crisis of misinformation, which is capable of precipitating the two aforementioned crises and, additionally, dividing society in order to frighten certain people's psyches.

As it turns out, addressing a military crisis that has morphed into an economic crisis is particularly tough at these times. Without a doubt, this can only be accomplished through the unity of society and the people, which is inextricably linked to the acceptance of accurate and expertly vetted information. Not only is there an issue with lying about whether a claim is true or untrue, but there is also a problem with a claim that cannot be verified at all. Verifying these claims is no longer straightforward and invariably requires the assistance of an expert. Fact-checking is the branch of expertise that results in the eradication of deception. While more debate regarding who fact-checks fact-checking specialists is permissible, there is little doubt that this promotes the elimination of disinformation.

Naturally, even fact-checking has its limitations. To describe the truth of a claim, one must first gather relevant facts about the claim, i.e. build a relevant knowledge base, and then use this knowledge base to write a lengthy text highlighting the claim's verification. In general , manual fact-checking is a time-consuming operation, with the ratio of claim origination to manual verification being substantially inversely proportional. This is where artificial intelligence or machine learning research comes into play. Without a doubt, the primary objective of research is to automatically check the veracity of claims with a high degree of precision and, preferably, to generate an explanation from fact-checking report.

However, such reports are lengthy, consisting of as many as tens of sentences. The authors of paper (Atanasova et al., 2020) recently demonstrated that a well-generated summary can increase computerized prediction of a claim's truthfulness. This is also related to the fact that a lengthy explanation may provide unimportant details regarding the claim, decreasing the reader's interest and hence the likelihood that the reader will read the explanation all the way through. Related to this, most critical information unfortunately is located at the end of a report. The authors of the well-known LIAR data set (W. Y. Wang, 2017) additionally include the final four sentences of explanation as the summary if no summary of explanation is provided.

As a result, we concentrated on developing a novel method for fact-checking explanation summaries in this work, whose contribution is as follows:

1. we scraped the latest data from Politifact website with summaries, making the data as clean as

possible (with motivation from paper (Raffel et al., 2019));

2. in our knowledge, we are the first to fine-tune the T5 transformer on fact-checking data and to propose a method of extractive summarization, which outperform the state-of-the-art results compared to article (Atanasova et al., 2020) and article (Kazemi et al., 2021);
3. we were the first to apply automatic summarization on Czech fact-checking data and thus verify the functionality on a multiligual T5 transformer model in the Czech language;
4. we finally tested the method on the well-known Czech news corpus SumeCzech (Straka et al., 2018), where we applied only 10 percent of the training data and improved several state-of-the-art results for the title generation.

Through the use of a randomly selected short fact-checking report, we first provide an overview of how machine learning via natural language processing contributes to automatic fact-checking (see Chapter 1), and then discuss various approaches to text processing and contextual representation of the fact-checking report's text (see Chapter 2). In Chapter 3 we demonstrate how to compare sentences in a particular fact-checking report using the *Local outlier factor*, which precedes Chapter 4, in which we cover three different approaches to text summarization and focus on the choice of the transformer model for abstract summarization. Then, in Chapter 5, we present our strategy for fine-tuning the SENTENCE-BERT model with the *Unified text to text transformer* (T5) to improve summarization. And in the last chapter 6, we compare our approach's performance on three different data sets.

1. Fact-Checking

1.1 Introduction

Due to the fast growth of *fake news* on social media and its detrimental impact on individuals and their general opinions (for example public votes in (Vosoughi et al., 2018)), a number of organizations are now manually fact-checking dubious claims. Indeed, fake news is often used to describe a variety of statements that are not necessarily linked to *veracity* or *truthfulness* (Vosoughi et al., 2018). The most obvious example of this type of use is when a „fake news“ is applied to media companies representing opposing political ideologies. Additionally, there is a relationship between fact-checking and *disinformation* or *misinformation*. While the first term implies deliberate intent to deceive the reader, the second term refers to the dissemination of information that may not be complete (Jowett et al., 2006). As a result of this inclusion, *disinformation* is seen as a subset of *misinformation*. While fact checking can assist in detecting misinformation, it cannot differentiate it from *disinformation*.

We view fact-checking as a critical component of journalism, as it enables us to accurately chronicle significant events by verifying *claim* or *statement*. The length of *claim* is one or few sentences asserting some facts. These facts are related with a category, alternatively termed *tag*, to which the accompanying *claim* belongs. The process of determining whether claims expressed in written or spoken language are truthful is known as *process of fact-checking*. This process is often undertaken by experienced professionals called *fact checkers*. Having a *claim*, they must review past speeches, conversations, laws, and published numbers or known facts, and utilize them, together with logic, to establish a label termed *veracity of claim*. Manual fact-checking cannot keep up with the speed and efficiency with which internet information is posted and distributed. Depending on the intricacy of the claim, fact-checking might take anything from an hour to several days (Hassan; Adair, et al., 2015).

1.2 Fact - checking report

The primary artifact produced by the fact-checking process is a *fact-checking report*, which is illustrated in Figure 1. Thus, the fact-checking begins by selecting claims that are generally relevant for fact-checking. The fact-checker next finds the necessary information elucidating the validity of the claim, referred as *ruling comments*, and determines the truth value labelled as *veracity*. To summarize ruling comments and to briefly highlight leading ideas of ruling comments , *justification* is created.

Politifact[1], Demagog[2], Factcheck[3], Fullfact[4], Snopes[5], Poynter[6], NewsGuard[7] are well-known initiatives that largely focus on fact-checking political claims. Since our work focus on Politifact[8] data (will be explain later), the report in Figure 1 was scraped from here. This report will be used as an example in following chapters as well and was purposefully picked because of its length (363 tokens) and content structure. As we can see the last sentence in Justification was deleted the same way as in the articles (Alhindi et al., 2018; Atanasova et al., 2020; Kazemi et al., 2021). We will discuss the cause in further detail in the section describing the data.

Claim: 15 days left to vote BUT if you are voting by mail, you need to vote TODAY. USPS says it needs a 14 day roundtrip to be counted on election day.

Ruling comments: The United States is expected to break records for voting by mail this year and that's creating a deluge of claims on social media about deadlines. Some are accurate, some are not."15 days left to vote BUT if you are voting by mail, you need to vote TODAY," reads one popular Instagram post. "USPS says it needs a 14 day roundtrip to be counted on election day." reads. This post was flagged as part of Facebook's efforts to combat false news and misinformation on its News Feed.(Read more about our partnership with Facebook.) The post is a screenshot of an Oct. 19 tweet by singer-songwriter Finneas Baird O'Connell, brother of singer Billie Eilish. Finneas' Twitter profile photo is of the Biden/Harris campaign logo, but he has no official role with the campaign. The post wrongly creates the impression that there is a national deadline to vote by mail. It's also a confusing message: the first sentence says voters must send it mail ballots 15 days ahead of time while the second sentence says the post office needs a 14-day roundtrip, suggesting that a voter can mail it in seven days ahead of time.
Here's what you should know:
The Postal Service did recommend in a national postcard in September that voters request the mail-in ballot at least 15 days before Election Day, Nov. 3, and return it at least seven days before Election Day. But this Instagram post omits the fact that states set their own laws about deadlines for receiving mail ballots. What's more, many states have options for voters to bypass the mail to return their ballots in an official ballot drop box or drop off site. Since the deadlines to return mail ballots vary by state, the best advice for voters is to check in with their local elections officials for information about when they must return their ballot, and their options for how to return it. Also, some states are automatically sending ballots to voters and therefore they don't have to request them. A spokesperson for the post office reiterated their previous advice, but also encouraged voters to check their state's requirements.

Justification: An Instagram post states "15 days left to vote BUT if you are voting by mail, you need to vote TODAY. USPS says it needs a 14 day roundtrip to be counted on election day. The post is unclear and omits important context. USPS said in a postcard in September that voters who want to have their ballots counted in the Nov. 3 general election should request the mail-in ballot at least 15 days before Election Day and return it at least seven days before Election Day. But states set deadlines for receiving mail ballots, and many jurisdictions allow voters to bypass the mail and return ballots in official ballot drop boxes or drop off sites. It's a good idea to return a ballot as soon as you can, but if you want to know the actual deadline for your state, check in with your state or local elections office. If you want to find out if your city or county has a place where you can drop it off, check in with your local elections office which typically posts that information on their website. ~~We rate this statement Half True~~.

Veracity: Half-True

Figure 1: Example instance of claim together with ruling comments, justification and its veracity

1.3 Automated fact-checking

As previously stated, fact-checking takes time, but require a great deal of information to conduct as well. Nevertheless, the effort to automate the various sub-processes of fact-checking increases the research interest in this field. The main purpose of automated fact checking is to lessen the human load associated with determining the truth of a claim. Recent advancements in the disciplines of *natural language*

[1]https://www.politifact.com/

[2]https://demagog.cz/, https://demagog.sk/, https://demagog.org.pl/

[3]https://www.factcheck.org/

[4]https://fullfact.org/

[5]https://www.snopes.com/

[6]https://www.poynter.org/

[7]https://www.newsguardtech.com/

[8]https://www.politifact.com/

processing (NLP), *information retrieval (IR)* have proved the feasibility of rapidly analyzing enormous amounts of textual material online, which proves automated fact-checking useful.

As seen in the Figure 2, the process of fact-checking may be far broader than we have previously indicated. Thus, we begin with *claim detection*, which entails selecting a credible claim and comparing it to previously fact-checked claims. If the claim is matched by evidence, then the *veracity* and *justification* should be given. If the claim is not matched, claim is *validated, verified*, and lastly, the veracity and justification are generated (see Figure 2).

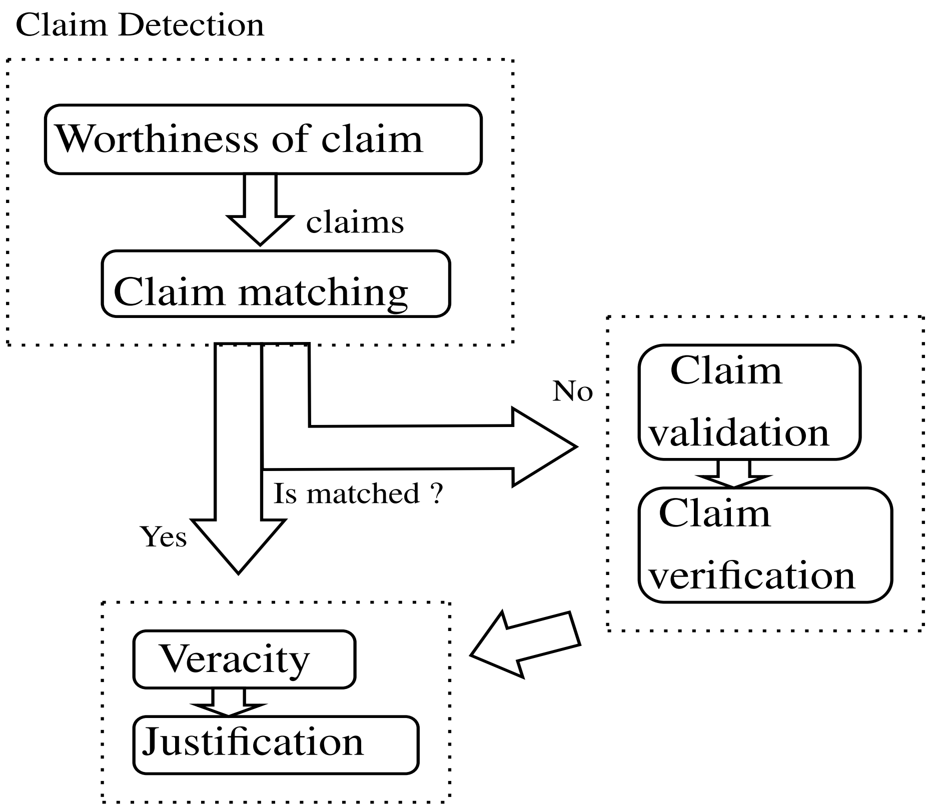


Figure 2: Automation of fact-checking

Restricting claims to those that are verifiable simplifies the automated process while minimizing the volume of information that requires manual fact-checking. This is called *claim worthiness*. For example, „I believe it is time to discuss the future" is not a claim that should be verified (Hassan; Arslan, et al., 2017), but we want to detect its relevancy automatically. To solve this, predicting factuality is usually a classification problem in which claims may be identified as requiring factual verification (Nakov et al., 2021).

Relatively new activity is *claim matching*, sometimes known as detecting already fact-checked claims. Claim matching for a claim detected in the claim detection component entails assessing if this is a claim that exists in the database and can be addressed by a prior fact-check. The job is written as follows: having a worthy claim and a database of previously fact-checked claims, it consists of finding if any of the claims in the database are connected to the input; at this point, the new claim does not need to be fact-checked again because it has already been fact-checked. Typically, it is structured as a ranking task, where claims in the database are ranked by similarity to query claim (Shaar et al., 2020).

Automated fact-checking also includes *claim validation*. This is the process through which a claim made in a certain context is assigned a truth value (Vlachos et al., 2014). To complete the process of claim validation, two primary techniques to verification (based on the assumption that their references are trustworthy) have emerged:

1. the claim is validated using existing knowledge sources (Shi et al., 2016);
2. the claim is validated using textual sources such as Wikipedia documents (Thorne et al., 2019).

As a result of *claim validation process*, *ruling comments* are generated. Those comments explain the veracity of the claim in details.

NLP researchers frequently handle *claim verification* as a text classification challenge. Having a claim under consideration and its collected evidence, models must determine if the claim is 'False,', 'True,' or 'Misleading.' In a more complex sense, veracity (truthfulness) of a claim, includes true, mostly-true, half-true, barely-true, untrue, or pants-on-fire label (W. Y. Wang, 2017; Hanselowski et al., 2019), whose prediction is more difficult to deal with through automated means and is sometimes compressed into fewer labels. The diversity in the sorts of labels employed by various research is an interesting point here. Some research works use truth values (e.g., true, false, and half-true) to determine a claim's veracity value. Others, on the other hand, relate to the idea of support (i.e., support and contradict), which determines if the claim and the reference agree.

Finally, producing *justification* for the veracity prediction is the newest research challenge in automated fact-checking. *Justification* is actually a concise summary of ruling comments regarding the veracity of claim. Researchers from article (Alhindi et al., 2018) found that providing a brief human explanation of the veracity label improved the system's performance by more than 10 % macro F1 score. To our knowledge, there were just two efforts at automating the process of creating explanations (Atanasova et al., 2020; Kazemi et al., 2021). Furthermore, extracting explanations offers information about places in the *ruling comments* that are similar to the gold justification, assisting the veracity prediction model in selecting the appropriate piece of evidence (Atanasova et al., 2020). Later on, an unsuccessful attempt was made to enhance initial and state-of-the-art work utilizing *GPT-2* (Kazemi et al., 2021). Hence, there is no doubt about the tendency to improve the automatic generation of *justification*.

2. From natural language processing to text similarity

This chapter discusses the process of converting a fact-checking report (text form) to *structured data* (numeric vectors). *Structured data* is data that has been developed in a quantitative manner. *Unstructured data*, on the other hand, refers to all other types of data that are not *structured data*, such as ruling comments of a fact-checking report.

As with the Report in Figure 1 from the previous chapter, the Table 1 shows the identical rulings comments in the first column. So in the first column we see ruling comments, which is truncated and the rows are merged just to save space. We will later present a variety of vectorization techniques for the sentences in the ruling comments. Because of that, we need to split these ruling comments into sentences. We prefer sentence-spliter[1] using python programming language to divide the ruling comments into sentences. The second column in Table 1 indicates the order of the sentences. The following column contains text of the sentences. The final column in the table interprets the sentences in their numeric form as vectors. Before we generate *contextual representations* of the these newly created sentences, various methods of *textual normalization* will be introduced.

Sentence vectors in the last column of Table 1 include elements which are called *weights*. These weights can be interpretable, i.e., directly related to the individual words of the sentences, in which case we call them *white box contextual representations*. This includes the vector representation of sentences using *term frequency - inverse document frequency (so called TF-IDF)*. In the other case, we talk about weights that are unintelligible and are dependent on neural network training such as WORD2VEC, DOC2VEC, BERT AND SENTENCE-BERT. We refer to these weights as *black box contextual representations* and will attempt to describe how this type of weights is generated.

[1]https://github.com/mediacloud/sentence-splitter

Ruling Comments	Order	Sentence	Sentence vector
	1	The United States is expected to ...	$[w_{1\,1}, w_{1\,2}, \dots, w_{1\,n}]$
The United States is expected to	2	Some are accurate, some are not.	$[w_{2\,1}, w_{2\,2}, \dots, w_{2\,n}]$
break records for voting mail this	3	...	...
year and that's creating a deluge of	4	...	...
claims on social media about	5	...	...
deadlines. Some are accurate,	6	...	...
some are not. 15 days left to vote	7	...	...
BUT if you are voting mail, you	8	Finneas' Twitter profile photo is of ...	$[w_{8\,1}, w_{8\,2}, \dots, w_{8\,n}]$
need to vote TODAY, reads one	9	...	...
popular Instagram post. USPS	10	...	...
says it needs a 14 day roundtrip to	11	...	...
be counted on election day. reads.	12	...	...
This post was flagged as part of	13	...	...
Facebook's efforts to combat false	14	...	...
news and ...	15	RELATED: Fact-checking the ...	$[w_{15\,1}, w_{15\,2}, \dots, w_{15\,n}]$
	16	A spokesperson for the post office ...	$[w_{16\,1}, w_{16\,2}, \dots, w_{16\,n}]$

Table 1: Chapter overview

DOC2VEC method of contextual representation is a modification of the WORD2VEC method and *Sentence Bert* is actually a modification of the *Bert* approach. Thus, generally, we will focus on the TF-IDF, WORD2VEC and BERT. Numerous articles have compared these three methods. For instance, the authors of the article (Salminen et al., 2020) compared these various contextual representations to several classifiers for hatred, concluding that BERT produced the best results. Similarly, the authors of the article (Carvallo et al., 2020) on *evidence-based medicine* compared the document representation using TF-IDF to that usingDOC2VEC and BERT on *active learning*, with BERT achieving the best results once again. Another fascinating project is the *classification of subjects* in a language other than English, Polish. TF-IDF, DOC2VEC and BERT were compared, and BERT again outperformed the others on practically all metrics (Walkowiak, 2021). The most pertinent study in terms of fact-checking is that in which the authors identify unifiable news articles for multi-document summarization (Singh et al., 2019). Unfortunately, only TF-IDF and DOC2VEC were applied here, BERT's method was only introduced and was being considered for future work.

Of course, there are various other contextual text representations, but the purpose of this chapter is to explain the most crucial changes that have occurred over time in the NLP field and to contrast them with the BERT method, which achieves the state of the art performance in wide variety of downstream tasks.

2.1 Bag of words

The easiest way to represent text data is through the use of a so-called *bag of words*. Assume we have a collection of sentences R, for example the ruling comments, which consist of m unique sentence s, denoted:

$$R = \{\mathrm{s}_1, \mathrm{s}_2, \mathrm{s}_3, ..., \mathrm{s}_m\}\,.$$

We will treat each sentence as just a set of words and will not consider their order at all (hence the name bag of words).

The corpus R can be represented as a matrix:

1. each row represents a single sentence,
2. there are as many columns as there are distinct words in the corpus, i.e. one column for each distinct word,
3. the i-th row (sentence $\mathrm{s_i}$) and j-th column (word $\mathrm{v_j}$) indicate the frequency with which the word $\mathrm{v_j}$ occurs in sentence $\mathrm{s_i}$.

For instance, let us divide the second sentence in the Table 1 into three sentences such that $R = \{\mathrm{s}_1, \mathrm{s}_2, \mathrm{s}_3\}$:

$$\begin{aligned} \mathrm{s}_1 &: \text{Some are accurate, some are not.} \\ \mathrm{s}_2 &: \text{Some are accurate.} \\ \mathrm{s}_3 &: \text{Some are not.} \end{aligned} \tag{1}$$

This corpus will be represented by a matrix shown in the following Table 2:

	some	are	accurate	not
s_1	2	2	1	1
s_2	1	1	1	0
s_3	1	1	0	1

Table 2: Bag of words example

This vector model of text has various weaknesses:

1. even when the collection of texts is somewhat small, the resulting matrix might be quite enormous and the quantity of separate words may be staggering,
2. while certain words (verbs, articles, conjunctions, and prepositions) will appear in nearly every sentence, non-traditional vocabulary may be considerably underrepresented,
3. sentence matrices and vectors are extremely sparse (they contain mostly zeros).

The following sections will discuss how to resolve or mitigate these issues through the use of suitable normalization or neural network training.

2.2 Text normalization

Generally, when performing *text analysis*, it is important to convert a text element to a numerical form that adequately analyzes the text and retains as much information as feasible. For now, let's stick to the sentence with the order of two in Table 1:

Some are accurate, some are not.

This sentence comprises even more compact textual elements than the sentence itself: *the tokens*. Splitting a sentence into tokens, or words, is frequently used as the principal method of text normalization in text processing. This process is referred to as *tokenization*. After applying *tokenization* this sentence will be splitted into list of tokens:

["Some", "are", "accurate", ",", "some", "are", "not", "."]

In this case, we could work with each word separately, for example counting the number of unique words in this sentence. However, the same words can of course be found in multiple sentences, but the problem is that the words may have different grammatical forms or be insignificant, i.e. „useless“. For example, look at the first word of the above sentence. The word „Some“ is in its basic form called *lemma*, but this word is meaningless because it can occurs in almost every single sentence of the ruling comments and therefore, this is not element that could potentially distinguish ruling comments sentences from each other. The second token is „are“ which is meaningless as well and its lemma is „be“. A collection of words that do not have any significant meaning are named *stop words*. Words which have same lemmas are having similar meaning and process of transferring tokens to its lemma is named *lemmatization*.

Lemmatization has many advantages such as:

1. reduce the number of unique words in a text limited to only lemmas,
2. since collection of stop-words has only lemmas included, lemmatization guarantees the removal of stop-words in all forms,
3. calculation of TF-IDF is reduced and performance of vectorization is improved (will be explained later).

So for demonstration, let us first lemmatise the above sentence:

["Some", "be", "accurate", ",", "some", "be", "not", "."].

where:	Stop-words	= „some“, „be“, „not“
	Punctuation	= „,“, „.“
	Significant word	= „accurate“

Finally, we remove unnecessary tokens of the sentences such as stop words and punctuation, where we get only one word representing whole sentence:

["accurate"].

2.3 Term frequency - inverse document frequency

*Term frequency - inverse document frequency (*TF-IDF*)* is a traditional word weighting technique used to interpret text. Returning to the sentences in ruling comments, the vector of each of these weightings (shown in Table 1) would have to contain the same number of elements, indicating the number of words in the vocabulary of the entire report. The position of each element of the vector would relate to a unique word. TF-IDF is the product of two numbers such as *term frequency* (TF) and *inverse document frequency* (IDF). The first of these numbers is *term frequency* (TF), which is a measure of how often a word occurs in a sentence (document). The usual ways of calculating (Salton et al., 1988):

$$\mathrm{TF}(\mathrm{v},\mathrm{s}) = \begin{cases} 1 & \text{if } \mathrm{v} \in \mathrm{s}, \\ 0 & \text{otherwise}, \end{cases} \tag{2}$$

$$\mathrm{TF}(\mathrm{v},\mathrm{s}) = \text{number of occurrences of word v in sentence s}, \tag{3}$$

where: v = word,
s = sentence.

The *inverse document frequency* (IDF) in comparison to TF in 2 or 3 is a measure that indicates how frequently or infrequently a word occurs within a corpus, for instance all sentences in the set R of ruling comments (see Table 1). The majority of calculation versions are based on the proportion of total sentences (denoted by $|R|$) and sentences containing the word v in equations 4 and 5:

$$\mathrm{IDF(v, R)} = \log\left(\frac{|R|}{|\{s \in R : \mathrm{v} \in \mathrm{s}\}|}\right) \tag{4}$$

$$\mathrm{IDF(v, R)} = \log\left(\frac{|R|}{1 + |\{s \in R : \mathrm{v} \in \mathrm{s}\}|}\right) \tag{5}$$

The second formula 5 is from the article (Kim et al., 2019), where *smoothing* is used to overcome the difficulty of division by zero. For word which is not current in sentence but presented in vocabulary, the IDF could not be calculated without number one in the fraction's denominator 5 because of division by zero.

Then, term frequency - inverse document frequency of word v included in sentence s is calculated:

$$\mathrm{TF-IDF(v,s)} = \mathrm{TF(v,s)} \times \mathrm{IDF}(\mathrm{v}, R) \tag{6}$$

.

Using the equation 3 and the equation 5, we can determine TD-IDF weights of the words from the set of sentences (see sentences in 1) in Table 3:

	some	are	accurate	not
s_1	0	0	0.30	0.30
s_2	0	0	0.30	0
s_3	0	0	0	0.30

Table 3: TD-IDF weights

2.4 WORD2VEC and DOC2VEC

In papers (Salminen et al., 2020; Carvallo et al., 2020; Walkowiak, 2021; Singh et al., 2019), the aim is to represent textual data by neural network models using the DOC2VEC *contextual representation* (Q. Le et al., 2014), which is a modification of the WORD2VEC *model* (Mikolov; Sutskever, et al., 2013). Therefore, we will first explain the training of a neural network by means of WORD2VEC *model* and then follow up with DOC2VEC *model*.

The architecture of the WORD2VEC *model* consists alternatively in the *Continuous bag of words method (CBOW)* or *Skip-Gram method*. The difference between these architectures is that *CBOW* using c words $\left(\mathrm{v}_{t-c}, \mathrm{v}_{t-(c-1)}, \ldots, \mathrm{v}_{t-1}\right)$ predicts a word v_t and *Skip-Gram* structure predicts a single word vector applied with vectors of concatenated word from the sequence $\left(\mathrm{v}_i, \mathrm{v}_{i+1}, \ldots, \mathrm{v}_{i+(c-1)}\right)$, where the parameter c is called a *window*. Each word will then be presented as vector $\boldsymbol{w}$. Additionally, we will determine the prescription of the elements of the given vector $\boldsymbol{w}$. This is important to clarify because in the next chapter 3, we will use these vectors to find outlier sentences in ruling comments. Therefore, we will draw from other articles where DOC2VEC and WORD2VEC *multilayer neural network* is explained in more details (Rong, 2014; Vargas-Calderón et al., 2019).

2.4.1 Single-neuron learning

The WORD2VEC and DOC2VEC models are based on a neural network. A neural network is a mathematical model that aims to behave similarly to a real neural network in the nervous system. An artificial neural network is a set of artificial neurons that have certain relationships between them. These relationships are given by values, called *weights*, which change with neural learning or training. Each neuron is given by an input vector $\boldsymbol{x} = (x_1, x_2, ..., x_D)$ and a vector of weights $\boldsymbol{w} = (w_1, w_2, ..., w_D)$, where $f\left(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}\right)$ is the scalar output and f is the so-called *activation function* (see Fig. 3).

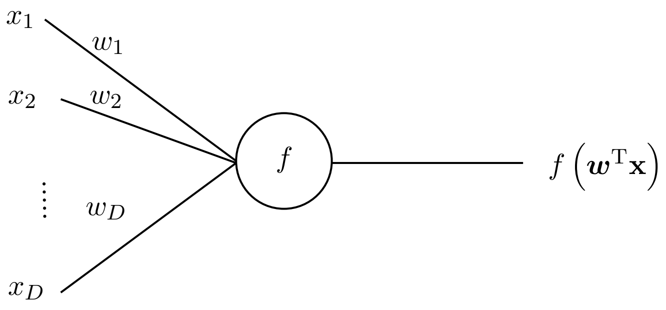


Figure 3: An artificial neuron

Assume

$$u = \boldsymbol{w}^{\mathrm{T}}\mathbf{x}. \tag{7}$$

Then, we apply the logistic function as the activation function, which we will also use in the following sections and define:

$$y = \sigma(u) = \frac{1}{1 + e^{-u}} \tag{8}$$

with important properties:

1. $\mathbf{y}$ takes values between 0 and 1,
2. $\mathbf{y}$ is continuous and smooth.

Thus, as we mentioned earlier, the weights of the $\boldsymbol{w}$ neuron are learned. By learning, we practically mean updating the values of the weights. In this example, we want them to learn so that the value of the output y gets as close as possible to the value of the actual output t. Therefore, let us give a suitable example of a *loss function (error function)*:

$$E = \frac{1}{2}(t - y)^2. \tag{9}$$

We want to **minimize** this *loss function* E by w_i for $w_i \; pre \; i \in \{1, 2, .., D\}$ and D is the number of elements of the input vector, then:

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial u} \cdot \frac{\partial u}{\partial w_i} \tag{10}$$

where the right-hand side (9) is multiplied by pairs of equal numerators with denominators and then:

$$\frac{\partial E}{\partial w_i} = (y - t) \cdot y(1 - y) \cdot x_i, \tag{11}$$

because $\frac{\partial E}{\partial y} = -1 \cdot (t - y)$, $\frac{\partial y}{\partial u} = y \cdot (1 - y)$ and $\frac{\partial u}{\partial w_i} = x_i$ (see eq. (7)). We thus apply stochastic gradient descent (Gardner, 1984) using the local optimum of the function (9):

$$\boldsymbol{w}^{(\mathrm{new})} = \boldsymbol{w}^{(\mathrm{old})} - \cdot (y - t) \cdot y(1 - y) \cdot \boldsymbol{x}, \tag{12}$$

where η is the learning rate constant, and $\eta > 0$ holds.

The initial weights are usually chosen randomly. These numbers keep updating recurrently during learning. By optimizing the loss function we obtain such elements $w_i \ for \ i \in \{1,2,..,D\}$ of vector $\boldsymbol{w}$, that approximate the output of neuron $\boldsymbol{u}$ to the correct value of $\boldsymbol{t}$.

2.4.2 Multi-layer neural network

Using a single neuron, we can get a structured weight vector $\boldsymbol{w}$ such that only the probability of a word determines the probability of the word independently of the words it is next to.
Therefore, let

$$\mathcal{R} = \{\mathrm{v}_k : k = 1,2,\ldots,R\} \tag{13}$$

is the vocabulary of the set of words in the whole corpus (for example, we consider unique words in ruling comments of Table 1), where v_k is the k-th word in the set $\mathcal{R}$. However, we now need a model that can predict the probability of a word conditionally given by another word. For this, we need a *multi-layer neural network* in which we first restrict ourselves to the input of one word and the output of another word. Specifically, the WORD2VEC tool uses a *three-layer neural network*, where the basis vector $\boldsymbol{x} \in \mathbb{R}^R$ is the input layer, the vector $\boldsymbol{h} \in \mathbb{R}^N$ is the hidden layer, and the vector $\boldsymbol{y} \in \mathbb{R}^R$ is the output layer (see Figure 4). Then we define the elements x_i of the input basis vector $\boldsymbol{x} \in \mathbb{R}^R$ (one-hot encoded vector) for the k-th word $\boldsymbol{v}_k$ shown in Fig. 4 as:

$$x_i = \begin{cases} 1 & i = k \\ 0 & i \neq k \end{cases} \tag{14}$$

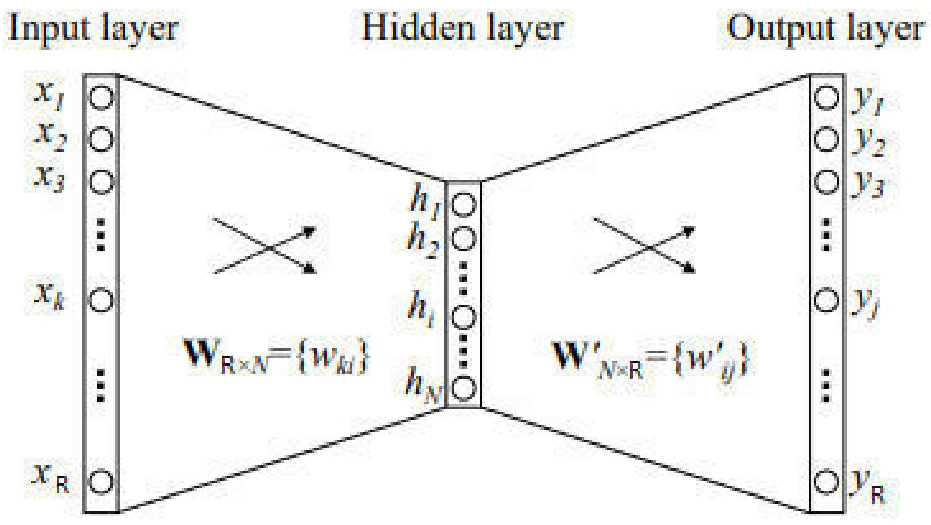


Figure 4: Three-layered neural network (Source edited from (Rong, 2014))

In our corpus, for the first word we would get an input vector $\boldsymbol{x}_1 = (1{,}0, \ldots, 0)$. We denote the weights in this network between the input and hidden layers by the matrix $\boldsymbol{W} \in \mathbb{R}^{R\times N}$. We then denote the k-th row of the matrix $\boldsymbol{W}$ as the vector $\boldsymbol{h}$:

$$\mathbf{h} = \boldsymbol{W}^T \boldsymbol{x} = \boldsymbol{W}^T_{(k,\cdot)} := \boldsymbol{v}_{\mathrm{v}_I^T}, \tag{15}$$

,where input word is interpreted as a vector $\boldsymbol{v}_{\mathrm{v}_I}^T$. This means that the relationship between the input layer and the hidden layer is linear (direct). Next, by applying a different weight matrix between the hidden and output layers $\boldsymbol{W'} \in \mathbb{R}^{R\times V}$ we compute the score u_j for each word in the word set $\mathcal{R}$ (def. in (13)):

$$u_j = \mathbf{v'}_{\mathrm{v}_j}^T \mathbf{h}, \tag{16}$$

where $\boldsymbol{v'}_{\mathrm{v}_j}^T$ is the j-th transposed column of the matrix $\boldsymbol{W'}$. We now use the score u_j as input to the soft-max function (a form of logistic regression) to find the conditional probability for multiple classes:

$$p\left(\mathrm{v}_j|\mathrm{v}_I\right) = y_j = \frac{\exp\left(u_j\right)}{\sum_{j'=1}^{R} \exp\left(u_{j'}\right)}, \tag{17}$$

where y_j is the output of the j-th neuron in the output layer and we want to maximize this probability. Thus, we show a function that is ideal, but is time-consuming because we need to compare each single word with all words in the set of words $\mathcal{R}$. Therefore, there are further refinements that select a score u_j only for such j words v_j, belonging to a subset of the set $\mathcal{R}$ so that a probability value very close to the probability value of this ideal function occurs. We will not discuss these refinements, because we want to explain how we obtain the final weights of the WORD2VEC *model*. The weights of the matrix $\boldsymbol{W}$ and the matrix $\boldsymbol{W'}$ are learned recursively similarly to the elements w of the vector $\boldsymbol{w}$ from subsection 2.4.1. However, the output matrix of the WORD2VEC model is the matrix $\boldsymbol{W} \in \mathbb{R}^{R\times N}$, whose k-th row vector $\boldsymbol{h} = \boldsymbol{v}_{\mathrm{v}_I}^T$ interprets the word $\mathrm{v}_k \in \mathcal{R}$.

2.4.3 Vector weights update in WORD2VEC model

This section will demonstrate how to obtain weights from the output layer to the hidden layer and then from the hidden layer to the input layer in a three-layer neural network such as **WORD2VEC**. We still continue to consider one word to be input and another one as output (see Fig. 4).

As mentioned in Section 2.4.2, we need to maximize the function 17, so we first explain the adjustment of the weights **from the output layer to the hidden layer** w_{ij} of the matrix W'. Then assume that v_O is the real word which we want to predict with an index in the output vector j^*. This word is conditioned by input word v_I. Then we want to maximize the *loss function* $E = -\log p\left(\mathrm{v}_O|\mathrm{v}_I\right)$ such that:

$$\begin{aligned} \max p\left(\mathrm{v}_O|\mathrm{v}_I\right) &= \max y_{j^*} \\ &= \max \log y_{j^*} \\ &= u_{j^*} - \log \sum_{j'=1}^{R} \exp\left(u_{j'}\right) := -E, \end{aligned} \tag{18}$$

which can also be thought as *entropy*, and we get this value with respect to the score, which comes from the random weights. Now, however, we define the change in the loss ∂E according to the change in the score ∂u_j, independently of the relation (18):

$$\frac{\partial E}{\partial u_j} = y_j - t_j := e_j, \tag{19}$$

where t_j is an element of the expected output, then we multiply by $\frac{\partial u_j}{\partial u_j}$:

$$\frac{\partial E}{\partial w'_{ij}} = \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial w'_{ij}} = e_j \cdot h_i \, . \tag{20}$$

Therefore, as in chap. 2.4.1 we apply *stochastic gradient descent* (Gardner, 1984) using the local optimum of the function E by w'_{ij}:

$$w'^{\,(\text{new})}_{ij} = w'^{\,(\text{old})}_{ij} - \eta \cdot e_j \cdot h_i \tag{21}$$

or

$$\boldsymbol{v}'^{\,(\text{new})}_{\text{v}_j} = \boldsymbol{v}'^{\,(\text{old})}_{\text{v}_j} - \eta \cdot e_j \cdot \boldsymbol{h} \text{ for } j \in \{1,2,\cdots,R\}\,, \tag{22}$$

where η is a learning constant, e_j is defined in terms of (19), h_i is the i-th element of the hidden layer $\boldsymbol{h}$, and $\boldsymbol{v}'_{\text{v}_j}$ is the output vector of the j-th word v_j. That is, the weights of the vector $\boldsymbol{v}'_{\text{v}_j}$ of the output word w_I are learned by subtracting the segments of the weights of the vector $\boldsymbol{h}$ (see (15)) of the input word w_O.

Practically speaking, these equations (21) a (22) mean that we have to go through every single word in the vocabulary $\mathcal{R}$ (def. v (13)) and check its probability of output y_j against the expected output t_j. The expected one-hot encoding output vector is the basis vector $\boldsymbol{t} \in \mathbb{R}^R$ with elements of t_j, similarly defined for the k-th output word in (14). This vector takes the value 1 at a different position than the input vector $\boldsymbol{x}$.

We can now use the matrix $\boldsymbol{W}'$ to explain the learning of the weights of the matrix $\boldsymbol{W}$ **from the hidden layer to the input layer**. First, we derive the *loss function* E according to the i-th element of the hidden layer $\boldsymbol{h}$:

$$\frac{\partial E}{\partial h_i} = \sum_{j=1}^{R} \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial h_i} = \sum_{j=1}^{R} e_j \cdot w'_{ij} := \text{EH}_i, \tag{23}$$

where u_j is the score of the j-th word from the output layer, $e_j = y_j - t_j$ is the prediction loss for the j-th word of the output. Then the vector $\boldsymbol{EH} \in \mathbb{R}^N$ with its elements EH_i is the sum of the output vectors of all the words in $\mathcal{R}$ (def. v (13)), which are weighted by the losses e_j. Further, by the expression (15), $h_i = \sum_{k=1}^{R} x_k \cdot w_{ki}$ necessarily holds, and we can derive the loss function E according to the vector of input weights (as in v (20)):

$$\frac{\partial E}{\partial w_{ki}} = \frac{\partial E}{\partial h_i} \cdot \frac{\partial h_i}{\partial w_{ki}} = \text{EH}_i \cdot x_k. \tag{24}$$

Since the vector $\boldsymbol{x}$ is input and basis with only one non-zero element, then only one row of the matrix $\frac{\partial E}{\partial \boldsymbol{W}} \in \mathbb{R}^{R \times N}$ is non-zero, let us denote it by $\boldsymbol{EH}^T$. This allows us to recursively define the learning of the elements of the matrix $\boldsymbol{W}$, as in the previous subsections, using a *stochastic gradient descent*:

$$\boldsymbol{v}^{(\text{new})}_{\text{v}_I} = \boldsymbol{v}^{(\text{old})}_{\text{v}_I} - \eta \boldsymbol{EH}^T, \tag{25}$$

where η is the learning constant and the row vector $\boldsymbol{v}_{\mathrm{v}_I}$ is the structured form of the input word v_I. So if we define a word as v_I, then his contextual representation would be row vector of weights $\boldsymbol{v}_I \in \boldsymbol{W}$.

For example, from the input data, the some words obviously occurs multiple times in the text, which is highly probable for frequent words. This means that we would have to learn the probability of this word based on other words. This is where neural networks have shown high efficiency by allowing us to teach them further. Thus, for example, we train the weights $\boldsymbol{W}$ for the first occurrence of the word v_I, store the weights as $\boldsymbol{W}_1$, and re-train and update these stored weights as $\boldsymbol{W}_2$, and so on. We train in this way until the last occurrence of v_I in the entire data. This whole cycle is called one **epoch**. To make the prediction even better, the probability of a word can be conditioned on the probability of multiple words, so in the next section we define the structure **CBOW - Continuous bag of words**.

2.4.4 CBOW - Matrix weights update in WORD2VEC model

As described in the introduction of section 2.4, the WORD2VEC tool may has *CBOW* or *Ski-gram* structure. These structures are different from the previous theory because we replace one word vector by multiple word vectors in the input neural layer *(CBOW)* or in the output neural layer *(Skip-gram)*. Because these two structures accomplish comparable outcomes empirically, we will focus exclusively on *CBOW* in this study, but the procedure of *Skip-gram method* is analogous.

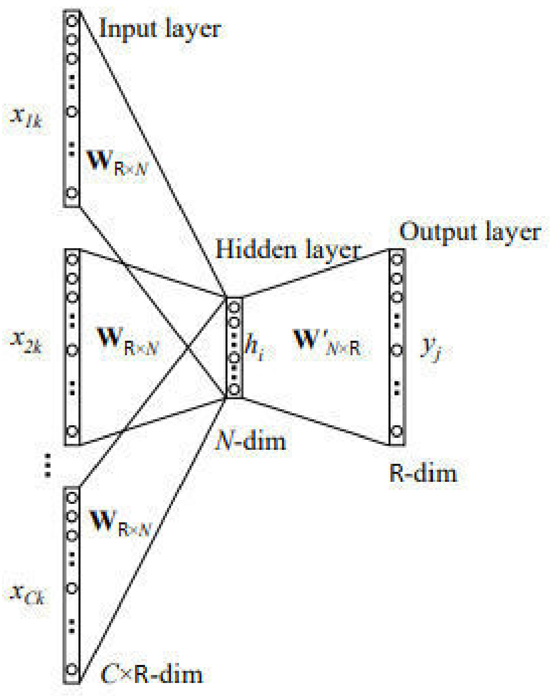


Figure 5: CBOW (Source edited from (Rong, 2014))

In the structure „CBOW“ we want to select multiple input vectors $\boldsymbol{x_c}$ for $c \in \{1,2,...,C\}$. Then, using the weight matrix $\boldsymbol{W}$, we get the input vector as the average of these vectors (see Figure 5):

$$\mathbf{h} = \frac{1}{C}\boldsymbol{W}^T(\mathbf{x}_1 + \mathbf{x}_2 + \cdots + \mathbf{x}_C) \tag{26}$$

$$= \frac{1}{C}(\boldsymbol{v}_{\mathrm{v}_1} + \boldsymbol{v}_{\mathrm{v}_2} + \cdots + \boldsymbol{v}_{\mathrm{v}_C})^T, \tag{27}$$

where $\mathrm{v}_1,\mathrm{v}_2,...,\mathrm{v}_C$ denote the corresponding words and word v has its input vector $\boldsymbol{v}_{\mathrm{v}}$ (see Fig. 5).

From this it is clear that we define the weights **between the hidden and the output layer $\boldsymbol{W'}$** similarly as in (22):

$$\boldsymbol{v}'^{(\text{new})}_{\mathrm{v}_j} = \boldsymbol{v}'^{(\text{old})}_{\mathrm{v}_j} - \eta \cdot e_j \cdot \boldsymbol{h} \text{ for } j \in \{1,2,\cdots,R\}, \tag{28}$$

but we define the row vector of weights **$\boldsymbol{W}$ between input and hidden** in one interaction for the c-th input word $\mathrm{w}_{I,c}$ as:

$$\boldsymbol{v}^{(\text{new})}_{\mathrm{w}_{I,c}} = \boldsymbol{v}^{(\text{old})}_{\mathrm{w}_{I,c}} - \frac{1}{C} \cdot \eta \cdot \boldsymbol{EH}^T \quad \text{for } c \in \{1,2,\cdots,C\}, \tag{29}$$

where η is a positive learning constant and the vector EH is defined by its elements in (23).

Now we can explain the updating of the input weights by simultaneously learning C row vectors of the matrix $\mathbf{W}$ using the I-th word v_I. The parameter C is *window* and the next parameter is N, which gives the number of columns of the weight matrix W, the number of rows of another weight matrix W', and the value of the hidden layer dimension $\boldsymbol{h}$ (see Fig. 5). We cannot determine the optimal value of the parameters C and N using WORD2VEC, but we know that WORD2VEC will determine the best values of the weights within any values of C and N.

2.4.5 DOC2VEC (SENTENCE2VEC)

In the previous section 2.4.4 we already discussed how to create weights for individual words in a text, which is an ultimate foundation for understanding how a neural network operates in natural language processing. However, this vector representation of individual words is ineffective for representing a sequence of words from the form of a sentence, a paragraph, or even a document. Therefore, paper (Mikolov; K. Chen, et al., 2013) was written in which the representation of a set of words was constructed as a concatenation of vectors of words or as an average of vectors of words, resulting in one vector representing a sentence.

However, it was discovered that adding an extra sentence vector or text vector is far more efficient (see for example original paper (Q. Le et al., 2014) or article with empirical comparison of DOC2VEC to WORD2VEC (Lau et al., 2016)). The SENTENCE2VEC or DOC2VEC method is an unsupervised learning algorithm for producing a vector representation of any arbitrary text, including sentence, paragraph or document. There are two different approaches for creating DOC2VEC vectors such as

Distributed Memory model and *Distributed bag of words*. Will again focus only on **Distributed Memory model** which is modification of *CBOW* which we discussed in section 2.4.4. Vectors in Distributed Memory model has been trained to predict the sentence's words. The DOC2VEC method, similarly to WORD2VEC, is used to generate a semantic vector for all sentences of a specified dimension. The advantage of this method over linear word vector combination is that it takes the order of the words in the document into consideration. Vectors will differ amongst documents that include the same words but in a different order. This technique use an extra document vector in comparison to technique explained in section 2.4.4. Each document's vector is unique, whereas word vectors are consistent across all documents.

Now, we can understand document as sentence and set of all sentences as ruling comments of those sentences. When creating a vector for a new (previously unseen) sentence, fixed word vectors are used to train the sentence vector. This vector is gradually updated with each iteration until it converges to the target solution *minimal and optimal loss function*. Prior to creating vectors for previously viewed sentences, the model must be trained constructing matrices $\boldsymbol{W}$ and $\boldsymbol{S}$. The matrix $\boldsymbol{W}$ includes the vectors of words, whereas the matrix $\boldsymbol{S}$ contains the vectors of sentences. Both of these matrices are initialized randomly at the beginning. The dimensions of the matrix $\boldsymbol{S}$ are $M \times N$, where M denotes the number of sentences in a set of ruling comments and N is the dimension of the hidden state. As seen in Figure 6, model training is comparable to the *CBOW* approach discussed in Section 2.4.4. The main difference is that here, in addition to the word matrix $\boldsymbol{S}$, a sentences matrix $\boldsymbol{W}$ is employed for word prediction, where both matrices $\boldsymbol{S}$ and $\boldsymbol{W}$ are updated separately.

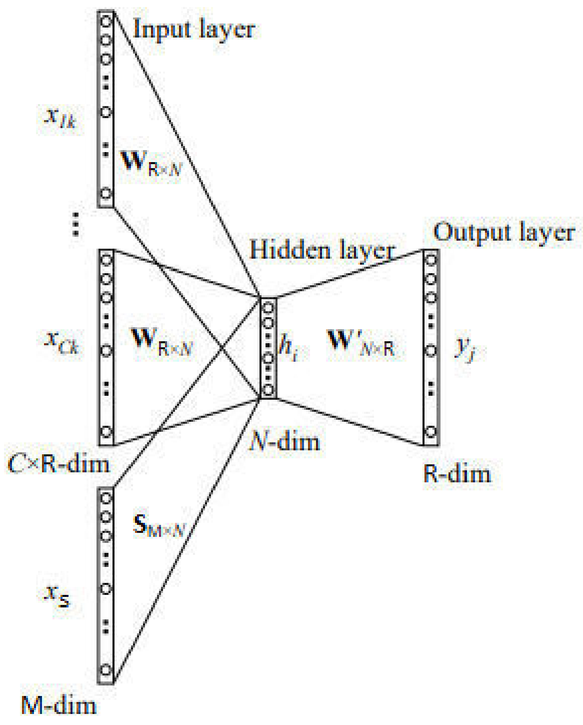


Figure 6: Distributed memory model (Source edited from (Rong, 2014))

Now, we have multiple one-hot encoding input vectors $\boldsymbol{x_c}$ for $c \in \{1,2,...,C\}$ and one-hot encoded sentence vector $\boldsymbol{x_s}$ for the s-th sentence in ruling comments, whose elements we define:

$$x_i = \begin{cases} 1 & i = s \\ 0 & i \neq s \end{cases} . \tag{30}$$

Then, using the weight matrix of words $\boldsymbol{W}$ and weight matrix of sentence $\boldsymbol{S}$, we get the hidden layer vector as the average of these vectors (see Figure 6):

$$\mathbf{h} = \frac{1}{C+1}\left(\boldsymbol{W}^T\left(\boldsymbol{x}_1 + \boldsymbol{x}_2 + \cdots + \boldsymbol{x}_C\right) + \boldsymbol{S}^T \boldsymbol{x_s}\right) \tag{31}$$

$$= \frac{1}{C+1}\left(\boldsymbol{v}_{\mathrm{v}_1} + \boldsymbol{v}_{\mathrm{v}_2} + \cdots + \boldsymbol{v}_{\mathrm{w}_C} + \boldsymbol{v}_s\right)^T , \tag{32}$$

where v_1,v_2, ...,v_C denote the corresponding words and word v has its related input vector $\boldsymbol{v}_{\mathrm{v}}$ and vector $\boldsymbol{v}_s$ is related to sentence where selected input words belong too (see Fig. 6).

Since we use average, we define the weights **between the hidden and the output layer $\boldsymbol{W}'$** similarly as in (22):

$$\boldsymbol{v}'^{(\text{new})}_{\mathrm{v}_j} = \boldsymbol{v}'^{(\text{old})}_{\mathrm{v}_j} - \eta \cdot e_j \cdot \boldsymbol{h} \text{ for } j \in \{1,2,\cdots,R\}\,. \tag{33}$$

and we define the row vector of weights $\boldsymbol{W}$ **between input and hidden** in one interaction for the c-th input word $\mathrm{w}_{I,c}$ as:

$$\boldsymbol{v}^{(\text{new})}_{\mathrm{v}_{I,c}} = \boldsymbol{v}^{(\text{old})}_{\mathrm{v}_{I,c}} - \frac{1}{C+1} \cdot \eta \cdot \boldsymbol{EH}^T \quad \text{for } c \in \{1,2,\cdots,C\}\,, \tag{34}$$

but row vector of sentence matrix $\boldsymbol{S}$ **between input and hidden layer** will be updated:

$$\boldsymbol{v}^{(\text{new})}_{s} = \boldsymbol{v}^{(\text{old})}_{s} - \frac{1}{C+1} \cdot \eta \cdot \mathbf{EH}^T \quad , \tag{35}$$

where η is a positive learning constant and the vector $\mathbf{EH}^T$ is same in equations (35), (34) and defined by its elements in (23).

2.5 TRANSFORMERS

In the introduction to this chapter, we demonstrated the contextual representations of sentences in a fact-checking report. We began with the ever-popular TF-IDF text representation, then went on to the neural network, where we described WORD2VEC and its variation, DOC2VEC, through the use of updated neural network weights. If the preceding neural contextual representations seem architecturally demanding, we will consider them as a simple basis to rely on in comparison to the complex transformer architectures. The effects of Transformer model can still be understood in the same way as for DOC2VEC or WORD2VEC.

A transformer is pre-trained on a massive amount of data, typically the entirety of Wikipedia (depending on the type of transformer). As a result, it is not applied to the downstream task by training the entire model from scratch. These models produce outstanding outcomes without the need for any kind of training or *fine-tuning* (zero shot classification, zero shot generation). However, *fine-tuning* of models using this architecture ensures significantly higher success rates than using only a basic, pre-trained transformer.

As previously stated, each word in WORD2VEC or TF-IDF was contextually represented by a single vector and each sentence was similarly contextually represented by a single vector (Doc2VEC). Individual words will be much more advanced in this section than they are in form of DOC2VEC, DOC2VEC, or TF-IDF vector, because each word is represented by up to three different types of vectors.

Transformer's primary contribution is the Attention mechanism (see article (Vaswani et al., 2017)), which we will discuss in subsection 2.5.2. Following that, we will cover the BERT *encoder* (see article (Devlin et al., 2018)), briefly discussing its pre-training but rather focusing on its *fine-tuning*, as in this study we will be *fine-tuning* this model rather than pre-training it.

2.5.1 Gentle introduction using Recurrent Neural Network (RNN)

If we recall the explanation of the sentence weights in the DOC2VEC model in section 2.4.5, every single sentence had exactly one response vector with all the words that occur in the sentence trained simultaneously. On the other hand, each unique word is often placed around different set of words which will ensure that this vector is updated multiple times. The process in which the weights of a word are updated according to the value of a loss function that is optimized using gradient descent is called **backpropagation** (see section).

Now, let us consider a sentence as an example that has been applied many times in this work:

$$\text{Some are accurate, some are not.} \tag{36}$$

In the DOC2VEC model we would update weights of this sentence and weights of the words of this sentence simultaneously. There is another approach, and that is **recurrent neural network**, which try to capture the dependencies between the individual words of this sentence, which in this case DOC2VEC does not do. Thus, by means of the *backpropagation* explained in section 2.4.1, we gradually update the weights of the sentence vector from the left side to the right side of this sentence, termed as **unidirectional recurrent neural network**. We are updating this sentence of this vector sequentially, where in each time unit one extra word is added as shown in Figure 7.

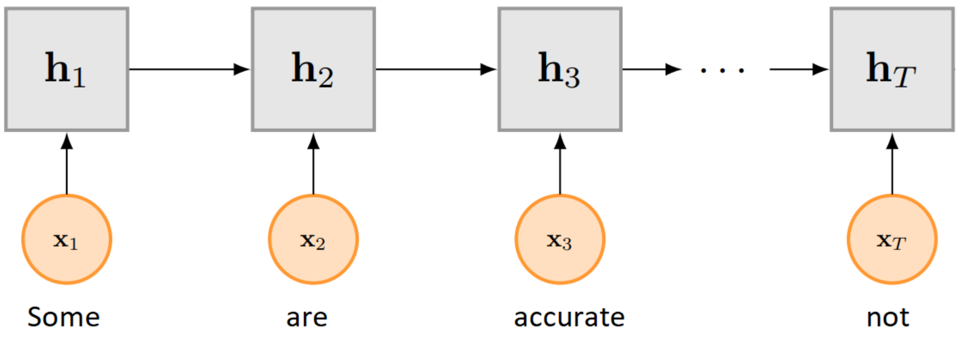


Figure 7: Recurrent Neural Network (Source edited from (Liu et al., 2016))

In Figure 7 we see a neural network with only one hidden layer $\boldsymbol{h}$, which is shown in a different time states. Each state of the hidden layer is updated by incrementally adding individual words to the sentence, where a change in word means a change in the time unit. Thus, in this particular case on Figure 7, the hidden layer is updated T times. The final layer $\boldsymbol{h}_T$ should represent the whole sentence given in 36.

By following article (Mikolov; Joulin, et al., 2014), we define one-hot encoding vector of current token as $\boldsymbol{x}_t$ and the output probability vector of next word $\boldsymbol{y}_t$. Assume sigmoid function σ (define in (8)). Then hidden layer $\boldsymbol{h}_t$ in time t is updated based in hidden layer $\boldsymbol{h}_{t-1}$ in time $t-1$ according to following:

$$\boldsymbol{h}_t = \sigma\left(\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{F}\boldsymbol{h}_{t-1}\right), \tag{37}$$

where $\boldsymbol{W}$ is word embedding matrix and $\boldsymbol{F}$ include recurrent weights. This is visualized in Figure 7. Then we can calculate output probability vector $\boldsymbol{y}_t$, according to the equation:

$$\boldsymbol{y}_t = f\left(\boldsymbol{W}'\boldsymbol{h}_t\right), \tag{38}$$

where f is soft-max function and $\boldsymbol{W}'$ is output matrix.

In the previous section 2.4 we very slightly mentioned gradient descent. Gradient Descent is a technique for finding the local minimum of a differentiable *loss function*. Of course this optimization algorithm is unavoidable also for *recurrent neural networks*.

Long short-term memory (LSTM) neural networks and *gated recurrent units (GRUs)* are the most dynamic recurrent neural networks, and have long been recognized as state-of-the-art models for language modeling and, in particular, machine translation. Recurrent neural networks with GRUs or LSTMs analyze input tokens sequentially and retain a state that describes the data viewed so far. This state is updated following the completion of each previous token. Thus, when the model processes the n-th token, it combines the state defining the sequence of the previous $\mathrm{n}-1$ tokens with information about the current token. This combination results in the creation of a state that represents the sequence of n tokens. The knowledge about prior tokens is theoretically capable of propagating throughout the entire token sequence. That is, after the n-th token is delivered to the input, the state formed comprises information about all prior tokens. In practice, this technique reveals flaws. By updating the weights of a *recurrent neural network* with high frequency, there are problems associated with gradient optimization (finding the local minimum of the function) such as problem of **exploding gradient** (Philipp et al., 2017) and **vanishing gradient problem** (Hochreiter, 1998). *Vanishing gradient* is much more complicated to solve than the exploding gradient problem. These problems can arise especially for *recurrent neural networks* that have a large number of steps.

Vanishing gradient

When the partial derivatives of the total error with respect to the weights of each neuron are **less than 1**, we have a vanishing gradient. When they are multiplied in the *backpropagation* process, the gradients used to update the weights have extremely small values. This means that the neural network will be unable to learn how to turn input to output correctly based on the training data. In the worst situation, a *vanishing gradient* might result in bringing training to a halt.

Exploding gradient

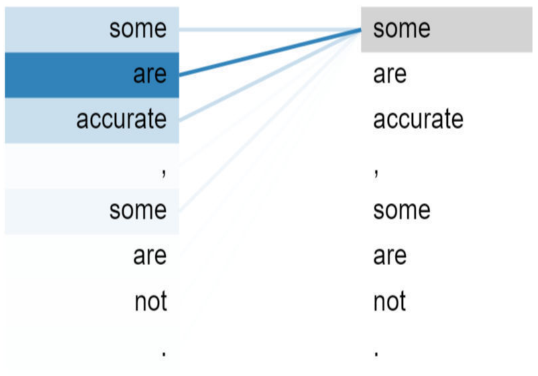
An *explosive gradient* can occur when the partial derivatives of the total error are bigger than one with regard to the weights of individual neurons. This results in significant fluctuations in the weights of individual neurons and the neural network is unstable. *Gradient clipping* is a general solution (Philipp et al., 2017). Gradient clipping is enabled by setting a threshold above which the gradient is clipped. This will keep the maximum change in the weights of neurons during training to a minimum.

To understand these problems practically, if we update the weights of the hidden layer of the sentence 36 illustrated in Figure 7 unidirectionally, from left to right for a very long sequence of words, then the information from the first word „Some“ starts to gradually disappear as time goes on. These issues were addressed by the introduction of the so-called **attention mechanism** (Vaswani et al., 2017). The "Attention" refers to the act of concentrating on a certain aspect of something and paying increased attention to that aspect. Additionally, the attention mechanism makes advantage of this principle and concentrates on specific elements when processing input data. All prior states are accessible to the *Attention layer*. It connects these states using unique relevance measures for the present token, providing significantly more accurate information about the relationship between tokens further out. The performance of *recurrent neural networks* was improved as a result of this technique. However, with the introduction of the transformer concept, it became evident that recurrent processing is unnecessary to achieve the same or better outcomes. Transformer processes all tokens concurrently and estimates the weights of each token's attention. In comparison to sequential token processing, this method enables efficient training on big data (Hassani et al., 2021; Zaheer et al., 2020).

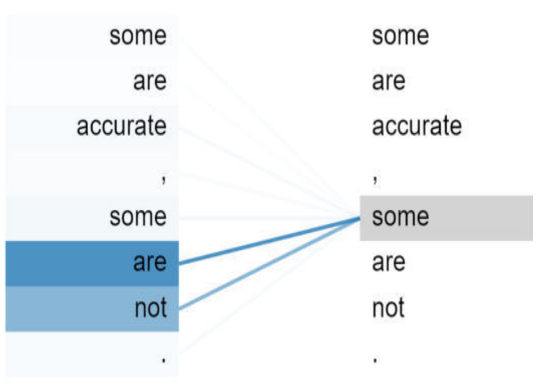
2.5.2 Attention

As previously stated, the primary contribution of Transformers is the **Attention mechanism** or **Attention layer** (Vaswani et al., 2017). This layer's objective is to capture the relationship between each word in a sentence and themselves. Thus, *attention* might be defined as the value placed on each individual word in relation to another word or to itself.

Now, let us reapply the sentence (36), which is depicted in Figure 8. We use BertViz[2] python library to visualize *attention mechanism* in this Figure. The darker line represents a higher attention value between the two words.



Attention to first token „some"



Attention to second token „some"

Figure 8: Attention mechanism for word „some" in different position of the sentence (36)

As illustrated in Figure 8, *attention* is varied for equal tokens placed in different positions inside a phrase. This is the primary distinction between *Transformer* and DOC2VEC or WORD2VEC. On the left hand side of Figure 8, we can observe that the first occurrence of the „are" token garners the most attention (relevance) to the first occurrence of token „some", followed by the similar *attention* of token „accurate" and token „some" itself. On the right side of Figure 8, the same token „some" has the largest significance to the second occurrence of „are" and a little lower *attention* to the token „not". As we can see, the token „accurate" has no meaning in this circumstance, despite the fact that it is the same word. This assures that the individual tokens in sentence can be interpreted by different vectors given by their position, in contrast to WORD2VEC (or DOC2VEC), which assigns each token a unique vector that does not alter based on its position.

Self-attention

The description of the detailed procedure of calculating Attention for each word in a sentence will be called **Self-attention**. By examining the level of the bold line in Figure 8, we can already somehow

[2]https://github.com/jessevig/bertviz

deduce what attention is. Of course, this is not completely detailed. The function of the Attention layer can be described as a mathematical operation on a triple of vectors query $\boldsymbol{q}$, key $\boldsymbol{k}$ and value $\boldsymbol{v}$. The output is given by the bound sum of the vector $\boldsymbol{v}$, where the weight assigned to each vector $\boldsymbol{v}$ is the scalar product of the vector $\boldsymbol{q}$ and the corresponding vector $\boldsymbol{k}$.

We will demonstrate how to compute the value of attention, but to calculate the attention value, we first require a sequence of tokens. A good illustration would be the tokens in the sentence (36), which are highlighted in Figure 8. As previously stated, this sentence comprises eight tokens. As a result, each token will eventually require eight attention values. We can define set of positions of each token as:

$$T = \{1, 2, \dots, 8\}. \tag{39}$$

Then, the function of the Attention layer can be described as a mathematical operation on a triple of vectors query $\boldsymbol{q}$, key $\boldsymbol{k}$ both of dimension d_k and value $\boldsymbol{v}$ with dimension of d_v. The output is given by the bound sum of the vector $\boldsymbol{v}$, where the weight assigned to each vector $\boldsymbol{v}$ is the scalar product of the vector $\boldsymbol{q}$ and the corresponding vector $\boldsymbol{k}$.

To explain this mathematically and in detail, for each token $i \in T$ (following (Vaswani et al., 2017)):

1. $\boldsymbol{x}_i \in \mathbb{R}^{d_m}$ is input embedding vector of i-th term and $\boldsymbol{W}^Q \in \mathbb{R}^{d_m \times d_k}$, $\boldsymbol{W}^K \in \mathbb{R}^{d_m \times d_k}$ and $\boldsymbol{W}^V \in \mathbb{R}^{d_m \times d_k}$ are weight matrices already trained (pre-training will be explained later):
$$\begin{aligned} \boldsymbol{q}_i &= \boldsymbol{x}_i \boldsymbol{W}^Q \\ \boldsymbol{k}_i &= \boldsymbol{x}_i \boldsymbol{W}^K \\ \boldsymbol{v}_i &= \boldsymbol{x}_i \boldsymbol{W}^V. \end{aligned} \tag{40}$$
2. As we computed those three kind of vectors, we need to get a score vector $\boldsymbol{s}_i$ for term i using dot product of same vector $\boldsymbol{q_i}$ sequentially against all key vectors $\boldsymbol{k_i}$ in the sentence, where elements of vector $\boldsymbol{s}_i$:
$$s_{i_j} = \boldsymbol{q}_i \boldsymbol{k}_j, \quad \forall j \in T. \tag{41}$$
3. This score has to be divided by dimension of the key vector d_k:
$$s'_{i_j} = \frac{s_{i_j}}{\sqrt{d_k}}, \quad \forall j \in T. \tag{42}$$
4. After we calculate the score vector $\boldsymbol{s}'_i$, we want to get sum of all elements in vector equal to one. So we apply softmax normalization:
$$s''_{i_j} = \frac{e^{s'_{i_j}}}{\sum_{j=1}^{8} e^{s'_{i_j}}}, \quad \forall j \in T. \tag{43}$$
5. We received vector of normalised score for i-th token $\boldsymbol{s}''_i$, which j-th element should multiple all values in vector $\boldsymbol{v}_j$:
$$\boldsymbol{v}'_{i_j} = s''_{i_j} \boldsymbol{v}_j, \quad \forall j \in T, \tag{44}$$
which means that we get multiplication of each value in vector $\boldsymbol{v}_j$ by the value of j-th element in the score vector $\boldsymbol{s}''_i$.

6. Finally, we sum up all $\boldsymbol{v}'_{i_j}$ vectors by elements:

$$\boldsymbol{z}_i = \sum_{j=1}^{8} \boldsymbol{v}'_{i_j}, \tag{45}$$

where we obtain a self-attention vector $\boldsymbol{z}_i$ whose j-th element represents the *self-attention* value of the i-th token to the j-th token. Thus, simply put, we explained the values that determine the coarseness of the lines between tokens in Figure 8.

To calculate all simultaneously, we can also define matrices exactly as in the paper (Vaswani et al., 2017):

$$\begin{aligned} \boldsymbol{Q} &\in \mathbb{R}^{8\times d_k} \\ \boldsymbol{K} &\in \mathbb{R}^{8\times d_k} \\ \boldsymbol{V} &\in \mathbb{R}^{8\times d_k}, \end{aligned} \tag{46}$$

where for the vectors in 40 hold:

$$\begin{aligned} \boldsymbol{q}_i &\in \boldsymbol{Q} \quad \forall i \in T, \\ \boldsymbol{k}_i &\in \boldsymbol{K} \quad \forall i \in T, \\ \boldsymbol{v}_i &\in \boldsymbol{V} \quad \forall i \in T. \end{aligned} \tag{47}$$

Then, we calculate output matrix such as:

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}}\right)\boldsymbol{V} \tag{48}$$

Multi-head attention

As explained in (Vaswani et al., 2017), rather than applying a single attention function on the d_m dimensional keys, values, and queries, it was found to be useful to linearly project the queries, keys, and values **h** times which is called **Multi-head attention**. Then we use different learnt linear projections:

$$\begin{aligned} \boldsymbol{W}_i^Q &\in \mathbb{R}^{d_m\times d_k} \\ \boldsymbol{W}_i^K &\in \mathbb{R}^{d_m\times d_k} \\ \boldsymbol{W}_i^V &\in \mathbb{R}^{d_m\times d_k}. \end{aligned} \tag{49}$$

Now remember the previous sections related to WORD2VEC or DOC2VEC, where the actual weight matrices $\boldsymbol{W}$ and $\boldsymbol{W}\prime$ are actually created by random weights. The matrices of the weights $\boldsymbol{W}_i^Q$, $\boldsymbol{W}_i^K$, $\boldsymbol{W}_i^V$ are also created with random weights. Therefore, to make these weight updates even more accurate leads to the same process of training these weights to be repeated, but with different values of the initial weights for i in $\{1, 2, \ldots, \mathbf{h}\}$. We again use sentence (36) as an instance with 8 tokens for which we want to create *Multi-head attention*. Thus, these three matrices are trained in the same way, with output matrices:

$$\boldsymbol{Z}_i \in \mathbb{R}^{8\times d_m}, \tag{50}$$

termed as „attention heads“. In each i-th *attention head* initial numerical weights of matrices $\boldsymbol{W}_i^Q$, $\boldsymbol{W}_i^K$, $\boldsymbol{W}_i^V$ are different. Then all matrices $\boldsymbol{Z}_i$ are concatenated into new matrix $\boldsymbol{Z}_{all} \in \mathbb{R}^{8h \times d_m}$:

$$\begin{aligned} \boldsymbol{Z}_{all} &= \text{CONCAT}\left(\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_i, \ldots, \boldsymbol{Z}_h\right), \\ \text{where } \boldsymbol{Z}_{\text{i}} &= \text{ Attention } \left(\boldsymbol{Q}\boldsymbol{W}_i^Q, \boldsymbol{K}\boldsymbol{W}_i^K, \boldsymbol{V}\boldsymbol{W}_i^V\right), \quad i = 1, \ldots, h. \end{aligned} \tag{51}$$

which is projected by matrix $\boldsymbol{W}^O \in \mathbb{R}^{d_m \times d_m}$ such that:

$$\text{MultiHead}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \boldsymbol{Z}_{all}\boldsymbol{W}^O \tag{52}$$

2.5.3 Bidirectional Encoder Representations from Transformers (BERT)

BERT is a model based on the Transformer architecture. Unlike *unidirectional models* (see RNN in section 2.5.1), which read the input text sequentially (left-to-right or right-to-left), the BERT model is **bidirectional** which reads the full sequence at once. This enables the model to provide context for the word depending on its context in the surrounding words. This property is used to perform model preprocessing on a text corpus without information from the teacher. After model is trained it can be fine-tuned on downstream task such as classification. Transformer is made up of two distinct mechanisms: an encoder that reads the input text and a decoder that generates a prediction for a particular input. Only the encoder mechanism is used in the BERT model. The encoder is constructed of six identical layers in the original concept. Each layer is divided into two sublayers. The first layer is a method for multi-head self-attention (explained in section 2.5.2). The second layer is a feed-forward neural network that is fully connected (as illustrated in Figure 9).

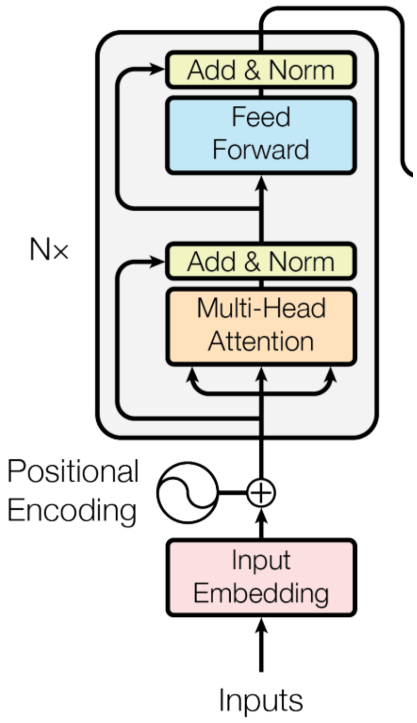


Figure 9: BERT architecture

Fully connected feed forward network

The multi-attention sub-output layer's is routed via a fully linked feed-forward network. This network is applied independently to each location in the token sequence. It consists of two transformations separated by a **ReLU** activation function (Vaswani et al., 2017):

$$FFN(x) = \max\left(0, xW_1\right) W_2. \tag{53}$$

where $\boldsymbol{x}$ is the output vector for a single token generated by the multi-headed attention sub-layer and $\boldsymbol{W}_1$ and $\boldsymbol{W}_2$ denote learnable parameters. The weights associated with these completely connected sublayers are not shared across encoder layers.

Normalization

BERT creates a residual dropout (Srivastava et al., 2014) around each of the two sublayers and then normalizes the layer outputs (Ba et al., 2016) such that the output of each sublayer:

$$\text{output}_{\text{res}} = x + \text{sublayer}(x) \tag{54}$$

where $\text{sublayer}(x)$ indicates the function that the sublayer implements.To ease these residual connections, the model's sublayers all provide outputs with the same dimension d_m.

BERT input representation

Given a series of words as inputs, which are limited to **maximum of 512 tokens**, BERT conducts a preliminary transformation on the words to generate numerical input representations for its model. In reality, these input representations are produced by **summing** three distinct forms of embeddings such as *token embedding, segment embedding, and positional embedding*. Figure 10 illustrates this input structure.

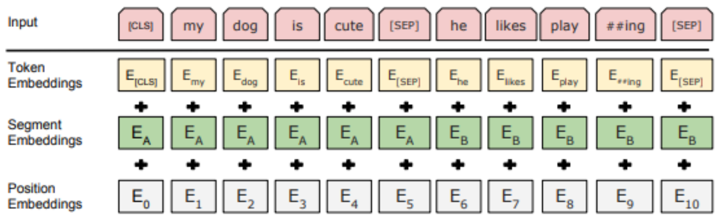


Figure 10: BERT input architecture

To clarify, we build an input vector $\boldsymbol{x}$ as a sum of those three kinds of embeddings, which we feed into the BERT attention mechanism as explained in subsection 2.5.2.

Token Embedding

As seen in Figure 10, the first embedding is a **token embedding**. In comparison to *Segment or Position embedding*, this kind of embedding is the most comparable to WORD2VEC. Preprocessing in BERT, on the other hand, is completely different since BERT is capable of handling *stop words* and *punctuations* as well. In BERT, tokenization is accomplished via the usage of **WordPiece tokenization** (Wu et al., 2016), which means that rather than deleting any words, words may be broken into many special tokens due to *WordPiece's fixed-size vocabulary*. During tokenization, the tokenizer first determines if a work exists in the vocabulary. If not, it has the capacity to break this term down into subwords from the dictionary and then breakdown it into individual characters. In Figure 10 (red blocks) you can see the split word „playing“ into two tokens „play“, „ing“. These tokens are created using the previously described *WordPiece Tokenization*. The token „ing“ is a kind of auxiliary token. After the word is tokenized into one or more *WordPiece subtokens*, the tokens' vocabulary ids are utilized to extract the appropriate token embedding vectors.

BERT's vocabulary includes the **30,000** most often occurring words and subwords in the English language, as well as all English characters with four unique tokens (Devlin et al., 2018):

1. [CLS], which serves as a unique classification token at the start of each sequence. For classification tasks, the final hidden state corresponding to this token is utilized as the aggregate sequence representation. It is disregarded in jobs that do not need categorization.
2. [SEP], which is used to denote the end of a sequence of sentence pairs. Additionally, it always

concludes the series.

3. [MASK], which is utilized in subsection 2.5.3 for the masked language modeling (MLM) training aim,
4. [PAD], as explained in article (Zhang et al., 2021), must be added to the end of the sentences to fill the empty slots in the input (if the length of the two concatenated sentences is shorter than the maximum sequence length, which is often length of 512 tokens in BERT).

Segment Embedding

As previously stated, another sort of embedding is segment-related (see Figure 10). The model must be able to determine if a given token belongs in sentence A or sentence B in BERT (recall that the model is trained by means of pairs of sentences). This is accomplished by creating additional segment embedding – one with a fixed token for sentence A and another for sentence B.

As illustrated in Table 4, we apply only two distinct segment vectors. The first sentence retains index 0 and the second sentence retains index 1. Thus, we use six identical vectors of index zero and another five identical vectors of index one. In other words, based on the amount of words in the first and second sentence, these two distinct vectors are replicated.

Tokens	[CLS]	my	dog	is	cute	[SEP]	he	likes	play	##ing	[SEP]
Index of segment embedding	0	0	0	0	0	0	1	1	1	1	1

Table 4: Segment embedding indexes

Positional Embedding

BERT, like the original TRANSFORMER, employs positional embeddings to inject information about the relative or absolute location of the tokens in the input sequence. These embeddings have the same dimension d_m as the *token and segment embeddings*, which simplifies their summarization. They are calculated with the help of *sine and cosine functions* of varying frequencies (Vaswani et al., 2017):

$$\begin{aligned} \mathrm{PE}_{(\mathrm{pos},\, 2i)} &= \sin\left(\frac{pos}{10000^{2i/d_\mathrm{m}}}\right) \\ \mathrm{PE}_{(\mathrm{pos}\,, 2i+1)} &= \cos\left(\frac{pos}{10000^{2i/d_\mathrm{m}}}\right) \end{aligned} \tag{55}$$

where pos denotes the coordinate system's location and i denotes the dimension. As a result, each dimension of the positional embedding is a *sinusoid*, and the wavelengths form a geometric progression from 2π to $10000 \cdot 2\pi$. Autors in (Vaswani et al., 2017) theorized that this function enables the model to readily learn relative locations, since PE_{pos+k} can be expressed as a linear function of PE_{pos} for any fixed offset k.

Input Embedding

Practically, we wish to get an input embedding vector $\boldsymbol{x}_v$ of a word v, which is later used for calculation of *attention* (see subsection 2.5.2). Because of that we must select three related vectors to token v, where $\boldsymbol{t}_v$ denotes the corresponding row vector of the *Token embedding matrix* $\boldsymbol{T} \in \mathbb{R}^{30000\times d_m}$, $\boldsymbol{s}_v$ denotes the corresponding row vector of the *Segment embedding matrix* $\boldsymbol{S} \in \mathbb{R}^{2\times d_m}$, and $\boldsymbol{p}_v$ denotes the corresponding row vector of the *Position embedding matrix* $\boldsymbol{P} \in \mathbb{R}^{512\times d_m}$. Then we compute the input embedding $\boldsymbol{x}_v \in \mathbb{R}^{d_m}$ for the attention mechanism as following:

$$\boldsymbol{x}_v = \boldsymbol{t}_v + \boldsymbol{s}_v + \boldsymbol{p}_v. \tag{56}$$

Pre-training BERT

BERT has been pre-trained on two tasks: *prediction of the next sentence (NSP)* and *masked language modeling (MLM)* (Devlin et al., 2018). The training loss is equal to the sum of the mean *MLM* and *NSP* likelihoods.

Prediction of Next Sentence

Next sentence prediction (NSP) is a binary classification problem in which the model fed pairs of sentences and trained to predict whether the second sentence in the pair is the following sentence in the original corpus (Devlin et al., 2018). This training objective assists in comprehending the relationship between pairs of sentences, which is not captured directly by language modeling (WORD2VEC or DOC2VEC) but is critical for a variety of downstream tasks such as *Question Answering (QA) or Natural Language Interference (NLI)*. This job is simply constructed using any monolingual corpus. To be more specific, while selecting the sentences A and B for each pre-training example, 50% of the time B is the actual sentence that follows A (labeled as „IsNext“), and 50% of the time it is a random phrase (labeled as „NotNext“). The final hidden vector corresponding to the [CLS] token is fed into a softmax output over the two possible predictions in this case. Here we receive probability of a sentences being „IsNext“ or „NotNext“.

Masked Language Modeling

Unlike *Continuous Bag of Words (CBOW)*, which seeks to predict the next word given the sequence of preceding words (see section 2.4.4),] *masked language modeling (MLM)* seeks to predict a percentage of randomly masked input tokens (Devlin et al., 2018). The *MLM* training objective was chosen over the standard *CBOW* aim due to BERT's bidirectionality (i.e., BERT predicts the target word using both left and right context in the sequence). Standard conditional language modeling cannot be utilized as a training objective for such models, as the bidirectional conditioning would allow each word to "see" itself indirectly, and the model would predict the target word trivially in a multi-layered context. While this enables the creation of a bidirectional pre-trained model, it introduces a mismatch between pre-training and fine-tuning, as the [MASK] token is not visible during fine-tuning. To address this, not all "masked" words are substituted with the actual [MASK] token. Because of that 15 percent of the tokens from the training corpus are changed by the following procedure:

1. in 80% of cases the token is replaced by a masking token [MASK],
2. in 10% of cases the word is the token replaced by a random word from the dictionary,
3. in 10% of cases the word is left unchanged.

Fine-tunning BERT

It is critical to highlight at this point that *fine-tuning* the BERT model entails retraining all the parameters end-to-end for a few epochs. Thus, as illustrated in Figure 11, while fine-tunning the model, we update all the weights of this transformer again, just as we do when training BERT from scratch, except that the data is frequent and very little in comparison to training from scratch.

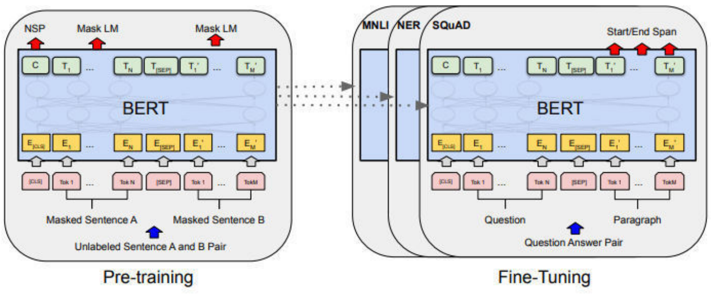


Figure 11: Pre-training and fine-tuning process for BERT

As illustrated in Figure 11, in order to train BERT from scratch, we require pairs of sentences, which is also similar to fine-tuning. At the input for fine-tunning, *sentence A* and *sentence B* shown in Figure 10 are equivalent to (Devlin et al., 2018):

1. sentence pairs in *paraphrasing*,
2. hypothesis-premise pairs in *entailment*,
3. question-passage pairs in *question answering*,
4. a degenerate text- pair in text classification or sequence tagging.

The token representations are fed into an output layer for token-level tasks like as sequence labeling or question answering, while the [CLS] representation is fed into an output layer for classification tasks such as entailment or sentiment analysis.

2.5.4 SENTENCE-BERT

The ordinary BERT architecture (explained in section 2.5.3) is suitable for a wide range of tasks like classification tasks, question answering and semantic sentence similarity. However, fine-tuning BERT end-to-end with all parameters of its neural network (Devlin et al., 2018) is highly time consuming

(Reimers et al., 2019). The BERT architecture is too complex to fine-tune and even if BERT is not fine-tuned, the **mean pooling** (average of all BERT tokens in sentence) achieves bad results (Reimers et al., 2019).

Given that the representation of BERT tokens reaches excellent results after *fine-tuning*, we need something further to it, allowing us to obtain comparable outcomes in sentence representation while the process of fine-tuning must be much quicker than in BERT. That is where SENTENCE-BERT with **Siamese Network** comes in. In the task of **textual similarity**, fine-tuning of SENTENCE-BERT by means of a Siamese neural network based on top of BERT definitely outperformed the results in comparison with the representation of sentences using *Universal Sentence Encoders, Vector of CLS token or Mean pooling of* BERT *tokens* (Reimers et al., 2019). Siamese network was first applied in Face recognition to improve clustering of images (Schroff et al., 2015) and later there was a great work in dialog systems for improving intent detection in multi class classification task (Ren et al., 2020). The latest relevant article (Behrendt et al., 2021) discuss fine-tuning of SENTENCE-BERT to improve embeddings for similarity of arguments.

More precisely, SENTENCE-BERT employs a pooling layer on top of each individual BERT instance, which implies that the encoder averages the collection of token embeddings provided in the BERT output into a fixed-length vector representing the sentence. Specifically, the pooling algorithm prefers either the [CLS] output token (bad peformance, see (Reimers et al., 2019)), *the maximum value of all word embeddings, or mean pooling of the ebeddings*. Additionally, the network is constructed of a layer that implements several objective functions, depending on the downstream task and the training dataset. There are multiple objective functions to fine-tune SENTENCE-BERT, but we will focus on **Triplet Loss objective function** (Reimers et al., 2019). Because experts in article apply always **cosine similarity** to compare similarity between two sentences (Reimers et al., 2019), we use it in **whole thesis** as distance metrics between a two vectors $\boldsymbol{u}$ and $\boldsymbol{z}$ and calculate as following:

$$d(\boldsymbol{u}, \boldsymbol{z}) = \cos(\boldsymbol{u}, \boldsymbol{z}) = \frac{\boldsymbol{u} \cdot \boldsymbol{z}}{\|\boldsymbol{u}\| \, \|\boldsymbol{z}\|}, \tag{57}$$

where $\|\cdot\|$ refers to l_2 normalization.

Triplet loss fine-tunning

SIAMESE BERT fine-tune less weights of neural network in comparison to BERT (Schroff et al., 2015). Now, suppose we have only three sentences:

$$\begin{aligned} \text{Anchor} &: \text{Some are accurate}, \\ \text{Positive} &: \text{Some are being accurate}, \\ \text{Negative} &: \text{Some are not accurate}, \end{aligned} \tag{58}$$

where their corespondent vectors will be $\boldsymbol{s}_a$ for anchor sentence, $\boldsymbol{s}_p$ for positive sentence and $\boldsymbol{s}_n$ for negative sentence. Then for receiving better representation of those vectors we have to train

SENTENCE-BERT by following objective function:

$$\max\left(d\left(\boldsymbol{s}_a-\boldsymbol{s}_p\right)-d\left(\boldsymbol{s}_a-\boldsymbol{s}_n\right)+\epsilon,0\right), \tag{59}$$

where d means *cosine distance* (57) and ϵ indicates the margin, which means that $\boldsymbol{s}_p$ is at least ϵ closer to $\boldsymbol{s}_a$ than $\boldsymbol{s}_n$.

As visualized in Figiure 12, we will input those three sentences into BERT independently to acquire token embeddings associated with the tokens in the sentences. *Mean-pooling* is used in our study to get sentence embeddings for each sentence, and those sentence embeddings are then fed into a **Siamese Network** that shares the same weights for each sentence embedding. During backpropagation, only those *Siamese Network weights* are updated with the goal of minimizing the cosine distance (57) between the anchor vector $\boldsymbol{s}_a$ and the positive vector $\boldsymbol{s}_p$ and maximising the distance between the anchor vector $\boldsymbol{s}_a$ and the negative vector $\boldsymbol{s}_n$.

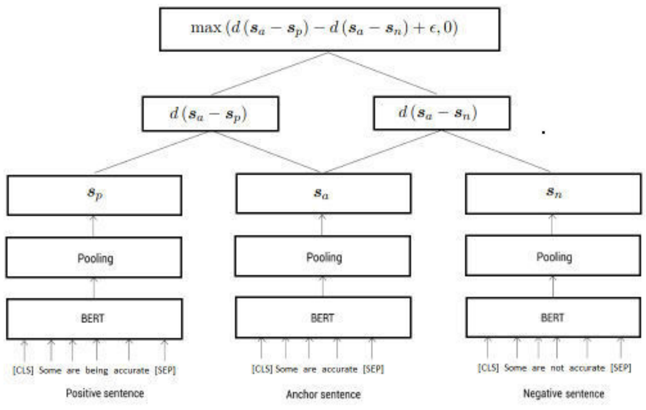

$\max\left(d\left(\boldsymbol{s}_a-\boldsymbol{s}_p\right)-d\left(\boldsymbol{s}_a-\boldsymbol{s}_n\right)+\epsilon,0\right)$
$d\left(\boldsymbol{s}_a-\boldsymbol{s}_p\right)$
$d\left(\boldsymbol{s}_a-\boldsymbol{s}_n\right)$
$\boldsymbol{s}_p$
$\boldsymbol{s}_a$
$\boldsymbol{s}_n$
Pooling
Pooling
Pooling
BERT
BERT
BERT
[CLS] Some are being accurate [SEP]
Positive sentence
[CLS] Some are accurate [SEP]
Anchor sentence
[CLS] Some are not accurate [SEP]
Negative sentence

Figure 12: Pre-training and fine-tuning process for BERT

Obviously, we'll fine-tune *Siamase Network* over numerous triplets, which will require limited GPU memory. Because of that, we fine-tune in **batches**. **Batch size** may be defined as the number of triples feed into a Neural network simultaneously.

3. Outlier Detection

In the previous chapter 2 we discussed various techniques to contextual encoding of sentences (TF-IDF, DOC2VEC, BERT), which allow us to convert a sentence in the form of a ruling comments into its numerical representation as vector. We usually use this contextual representation in vectors to be able to compare how similar a pair of two sentences are to each other. Thus, we can determine the degree of similarity between any two sentences using several similarity metrics (see the end of the chapter 2).

If we wish to reconsider the degree to which each sentence is similar to multiple others, a measure such as *cosine distance* alone is insufficient. Typically, we would assume some binary truth value, indicating whether a particular sentence is comparable to a collection of other sentences, termed an *inlier*, or is completely dissimilar to those sentences, dubbed an *outlier*. As a result, it is more meaningful to give a degree of „outlierdom" to each sentence. This degree is referred to as the sentence's, text's, or object's **local outlier factor** (LOF). It is local in the sense that the degree of isolation depends on how isolated the sentences are in relation to their immediate neighborhood of other sentences (Breunig et al., 2000).

3.1 Local Outlier Factor

Local outlier factor, called LOF, is an outlier detection algorithm (anomaly detection), which we classify as a density-based algorithm (Breunig et al., 2000). This algorithm is also a modification of the nearest neighbor algorithm, henceforth NN, which leads to the fact that the detection by LOF does not only depend on the density of the sentence neighborhood, but also on the distance of the sentences (we can still think about sentences in a ruling comments). Thus, the main difference between LOF and NN is that LOF computes the relative density of each sentence of a ruling comments, and hence we classify it as a density-based distance algorithm. Compared to LOF, NN computes the sum of the distance to each neighbor sentence.

There are numerous outlier identification methods, but when working with text, the application of LOF produces the best results. For instance, in the study (Seo et al., 2020), the authors attempt to detect uncommon customer views using DOC2VEC and LOF, vectorizing all customer opinions in the corpus first and then unusual opinions are identified from which significant keywords and phrases are extracted. It is necessary to mention the most recent work (Jeon et al., 2022), which follows a very similar approach for determining the novelty of patents, beginning with the use of DOC2VEC to produce a patent vector for each patent and ending with the use of LOF to create a local outlier value for the new patent. Another area of investigation is the application of LOF to aspect based sentiment analysis and the evaluation of spam detection (You et al., 2020). Similarly, the LOF has demonstrated selective outcomes on structured data (not text) in the detection of photovoltaic system faults (Ding et al., 2018).

In the following definitions, we consider ruling comments and **16** sentences only in their vector form from Table 1 as an example. Then, let $\boldsymbol{S} \in \mathbb{R}^{16\times n}$ be the matrix vector representations shown in the last column of the Table 1 and $\boldsymbol{s}_i$ to be i-th row vector of i-th sentence (transformation of a vector from text is discussed in previous chapter 2). Then we define matrix $\boldsymbol{S} \in \mathbb{R}^{16\times n}$ such that:

$$\boldsymbol{S} = \begin{bmatrix} \boldsymbol{s_1} & \boldsymbol{s_2} & \cdots & \boldsymbol{s_i} & \dots & \boldsymbol{s_{16}} \end{bmatrix} \tag{60}$$

To explain the LOF algorithm, let us first define the **k-distance of a sentence** in general as follows:

Definition 1. *Let $k \in \mathbb{N}$, $\boldsymbol{S} \in \mathbb{R}^{16\times n}$ is set of sentences,*
$d(\boldsymbol{p}, \boldsymbol{q})$ is cosine distance between two sentence vectors $\boldsymbol{p} \in \boldsymbol{S}$ and $\boldsymbol{q} \in \boldsymbol{S}$ and it holds:

(a) for at least k vectors $\boldsymbol{q\prime} \in \boldsymbol{S} \setminus \{\boldsymbol{p}\}$, such that $d(\boldsymbol{p}, \boldsymbol{q\prime}) \leq d(\boldsymbol{p}, \boldsymbol{q})$
and
(b) for at least $k - 1$ sentences $\boldsymbol{q\prime} \in \boldsymbol{S} \setminus \{\boldsymbol{p}\}$, such that $d(\boldsymbol{p}, \boldsymbol{q}') < d(\boldsymbol{p}, \boldsymbol{q})$,

*then we define $\text{DIST}_k(\boldsymbol{p}) = d(\boldsymbol{p},\boldsymbol{q})$ and we call this value the **k-distance of sentence $\boldsymbol{p}$**.*

So, in other words, the *k-distance of sentence $\boldsymbol{p}$* interpreted by the row vector of the sentence matrix $\boldsymbol{S}$ (defined in Equation (60)) is computed as the k-th smallest *cosine distance*. Using the k-distance of the sentence vector $\boldsymbol{p}$, we can define the neighborhood of the k-distance of the conversation $\boldsymbol{p}$ as follows:

Definition 2. Neighborhood of a sentence vector $\boldsymbol{p}$ with radius k *is the set*

$$N_k(\boldsymbol{p}) = \{\boldsymbol{q} \in \boldsymbol{S} \setminus \{\boldsymbol{p}\} \mid d(\boldsymbol{p},\boldsymbol{q}) \leq \text{DIST}_k(\boldsymbol{p})\}. \tag{61}$$

These sentence vectors $\boldsymbol{q}$ are elements of the *k-remote neighborhood $\boldsymbol{p}$*. That is, the k-distance neighborhood for some sentence vector $\boldsymbol{p}$ is the neighborhood of those vectors $\boldsymbol{q}$ that have *cosine distance* at most equal to the *k-distance of the sentence $\boldsymbol{p}$* and the set of these vectors does not include the vector $\boldsymbol{p}$ itself. Next, we introduce the term *Reachability distance*, which is the between the distance of the sentence vector $\boldsymbol{p}$ from the sentence vector $\boldsymbol{q}$ and the k-distance of the sentence vector $\boldsymbol{p}$:

Definition 3. Reachability distance *of a sentence vector **q** from a sentence vector **p** by the k-th neighbor is defined by the relation*

$$\text{REACHDIST}_k(\boldsymbol{p}, \boldsymbol{q}) = \max\left\{\text{DIST}_k(\boldsymbol{p}), d(\boldsymbol{p}, \boldsymbol{q})\right\}. \tag{62}$$

Thus, $\text{REACHDIST}_k(\boldsymbol{p}, \boldsymbol{q})$ for a given sentence vector $\boldsymbol{q}$ is equal to the maximum value between the k-th lowest distance of the sentence vector $\boldsymbol{p}$ and the distance of the sentence vector $\boldsymbol{p}$ from $\boldsymbol{q}$. Therefore, we must note that if the sentence vector $\boldsymbol{p}$ is far away from the sentence vector $\boldsymbol{q}$, then their $\text{REACHDIST}_k(\boldsymbol{p}, \boldsymbol{q})$ is equal to their own distance, otherwise of course equal to the k-th lowest distance of the sentence vector $\boldsymbol{p}$.

Based on the reachability distance of the sentence vector $\boldsymbol{q}$ from the sentence vector $\boldsymbol{p}$ (defined by the expression (62)) and the neighborhood of the sentence vector $\boldsymbol{p}$ with radius k (def. by the expression (61)), the *local reachability density* is computed as:

Definition 4. *Local achievable sentence vector density* $\boldsymbol{p}$ *is*

$$lrd_k(\boldsymbol{p}) = \frac{\mid N_k(\boldsymbol{p}) \mid}{\sum_{\boldsymbol{q} \in N_k(\boldsymbol{p})} \text{REACHDIST}_k(\boldsymbol{p}, \boldsymbol{q})}. \tag{63}$$

Again, in practical terms of this thesis, this means that the expression (63), the local reachability density of the sentence vector $\boldsymbol{p}$, is equal to the inverse of the average reachability distance of those sentence vectors from the sentence $\boldsymbol{p}$ that belong to the set of elements of the k-distant neighborhood of the sentence vector $\boldsymbol{p}$. With the help of the local reachable density, we can thus finitely compute the *local outlier factor* LOF for the sentence vector $\boldsymbol{p}$ as:

Definition 5. *Local outlier factor of sentence vector* $\mathbf{p}$

$$LOF_k(\boldsymbol{p}) = \frac{\frac{1}{N_k(\boldsymbol{p})} \sum_{\boldsymbol{q} \in N_k(\boldsymbol{p})} lrd_k(\boldsymbol{q})}{lrd_k(\boldsymbol{p})}, \tag{64}$$

where $LOF_k(\boldsymbol{p}) \in \mathbb{R}^+$.

Roughly said, if sum of all $lrd_k(\boldsymbol{q})$ such that $\boldsymbol{q} \in N_k(\boldsymbol{p})$ increase, than LOF_k for sentence $\boldsymbol{p}$ will be increasing and if $lrd_k(\boldsymbol{q})$ increase, then LOF will be decreasing and vice versa.

Since the local outlier factor $LOF_k(\boldsymbol{p})$ can reach any positive real number, we decided to normalize it for the purpose of this thesis:

Definition 6. *Local outlier factor of sentence vector* $\mathbf{p}$ *normalized*

$$LOF_k(\boldsymbol{p})_{Norm} = 1 - \frac{1}{1 + LOF_k(\boldsymbol{p})}, \tag{65}$$

where $LOF_k(\boldsymbol{p})_{Norm} \in (\,0{,}1\,)$.

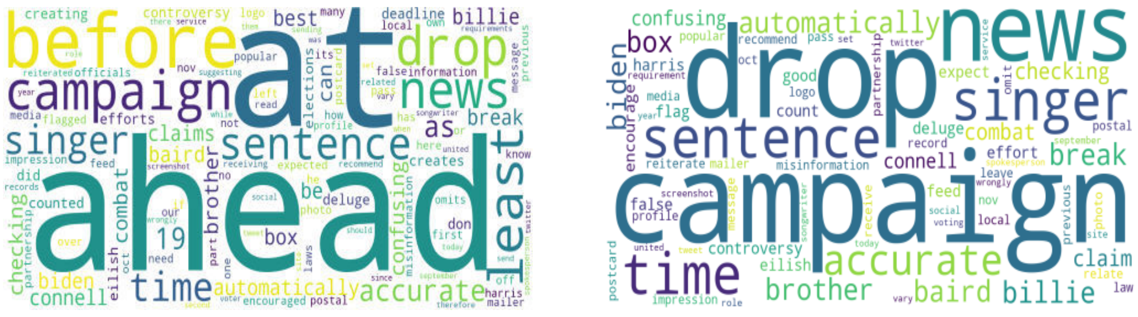
3.2 LOF application on a random report

As with the preceding text, this section will concentrate on the Report 1, whose ruling comments are illustrated in the Table 1. Of course, any other report may be served as an example. We will first visualize the sentences in this report in terms of textual normalization (wordclouds), and then demonstrate how we may graphically compare the *Local outlier factor* of the sentences included in this report. This part assumes a working knowledge of the contextual representation of a text or sentence in vector form and text normalization (see chapter 2) and a basic knowledge of the *Local outlier factor* introduced in Section 3.1.

By text normalization, we simply mean the elimination of stop words, lemmatization, etc. (see section 2.2). Now, we display the individual word counts using a wordcloud[1], where the coarser and larger words represent the more often occurring words in the sentences contained within a certain report.

[1]https://pypi.org/project/wordcloud/

When we compare the left wordcloud against the right wordcloud in Figure 13, we can clearly see that the most frequent words in the left wordcloud with only one occurrence in a sentence are those that have no meaning for the description of a sentence in the report („ahead“,„at“,„before“, etc.). On the other hand, by means of normalization we reduce the total number of words in the vocabulary and we get words that can be really meaningful and appear at least once but always in the same sentence („campaign“,„drop“,„news“, etc.). By applying normalization, the number of unique words appearing in one sentence is reduced from **122** to **74**.

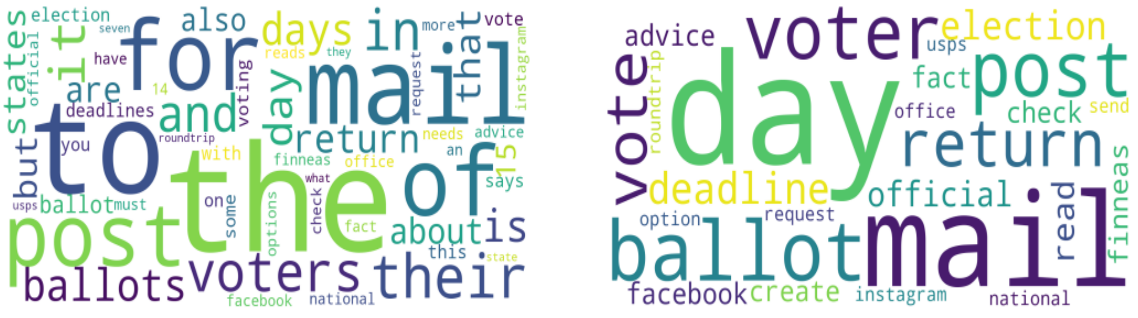


Without text normalization (122 tokens) With text normalization (74 tokens)

Figure 13: Words occurring in only one sentence of the report

Then again, not surprisingly, if we focus on words that are found in multiple sentences, we get a bunch of meaningless words with the highest frequency (see wordcloud on the left in Figure 14). The wordcloud on the right in Figure 14 is fascinating because normalization reduces the amount of words in the vocabulary by more than half, from **58** words to **25** words. This selects those words that can add meaning to the vector representation of the sentence (see the wordcloud on the right in Figure 14).

In the following subsections we will show how text normalization is actually related to contextual representation using TF-IDF, DOC2VEC AND BERT in order to visualize LOF for individual sentences of the report.



Without text normalization (58 tokens) With text normalization (25 tokens)

Figure 14: Words occurring in more than one sentence of the report

3.2.1 LOF based on the TF-IDF, DOC2VEC or SENTENCE-BERT sentence vectors

Before we applied LOF on TF-IDF (or DOC2VEC) vectors, every sentence in Figure 15 has been normalized textually. This is a standard approach when using TF-IDF (or DOC2VEC), particularly for very tiny data sets (report sentences). In contrast, SENTENCE-BERT model does not require any kind of textual normalization because it was trained on huge amount of not-textually-normalized data.

Recall that the larger the LOF value, the more distinct the sentence actually is from the other sentences in the report. There is no doubt, when looking at blue points in the graph 15, that the second sentence achieves the highest LOF value because the sentence **„Some are accurate, some are not.“** in no way affects the accuracy of the report and is meaningless at all.

Focusing on the DOC2VEC model in graph 15, the distribution of green points is very chaotic compared to TF-IDF. The problem of DOC2VEC, based on this one report , seems to be the insufficient number of sentences in the report, which may not be the same for longer reports (with more sentences). We believe that definitely second sentence of the report should be the biggest outlier because no significant information is provided here. In case of DOC2VEC, however , there are additional sentences that reach even higher LOF than the second sentence (see Figure 15).

SENTENCE-BERT vectorization of the sentences is visualized by red points in the graph 15. Thus, we apply the SENTENCE-BERT model to all words in the given report without removing any of them. This means that we bind the words from the wordcloud on the left side of the Figure 13 and on the left side of the Figure 14. We can only afford to do this if the neural network is pre-trained on a gigantic corpus of plain text data. This is of course not the case for TF-IDF and certainly not DOC2VEC, since we train these models on just a few sentences of the example report above.

The results of SENTENCE-BERT are shown by red points in Figure 15. Even we do not apply the textual normalization in the Figure, we can see that by SENTENCE-BERT the second sentence reach LOF over 0.65 in comparison to nearly 0.53 for nnLOF in TF-IDF (see blue points in 15).

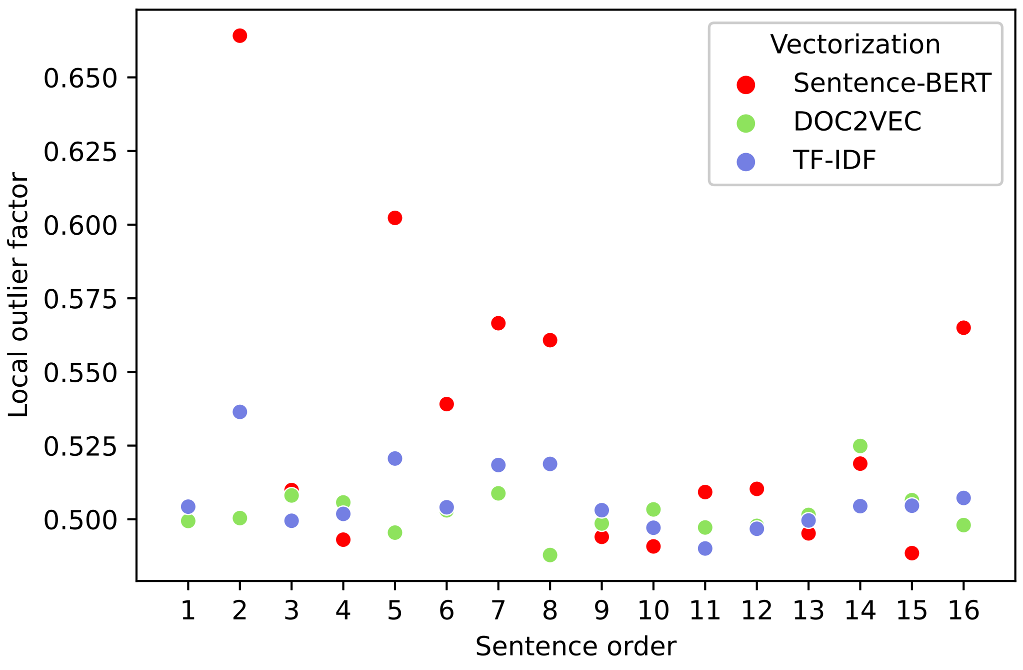


Figure 15: LOF of the sentences in the report

4. Automatic text summarization

In this chapter we will explain what automatic summarization is, what are the three main types of automatic summarization and finally we will focus on the model that was chosen for this work.

A summary of a text may be broadly defined as the extraction of essential information from a longer text with the primary goal of reducing the text's size while retaining the text's major ideas. This procedure is often carried out manually by humans, which results in high-quality summaries. However, creating such summaries is very time intensive, much alone the frequent manual summarizing, which may also be psychologically demanding for humans.

There are other types of summary (El-Kassas et al., 2021; Alomari et al., 2022), but the purpose of this work is to distinguish between **extractive** and **abstractive** summarization, as well as the mixture of the two known as **hybrid** summarization.

4.1 Extractive text summarization

Extractive summarization extracts a set of multiple word sequences from the input content that must be specifically discovered in the text. Figure 16 illustrates the architecture of the extractive text summarization system, which consists of the following steps (El-Kassas et al., 2021):

1. pre-processing the input text via some form of text normalization (see section 2.2);
2. post-processing, such as reordering the extracted phrases and concatenate them;
3. the processing, which entails building a contextual representation of the document's subsequences, ranking them and delete certain subsequences by the ranking score.

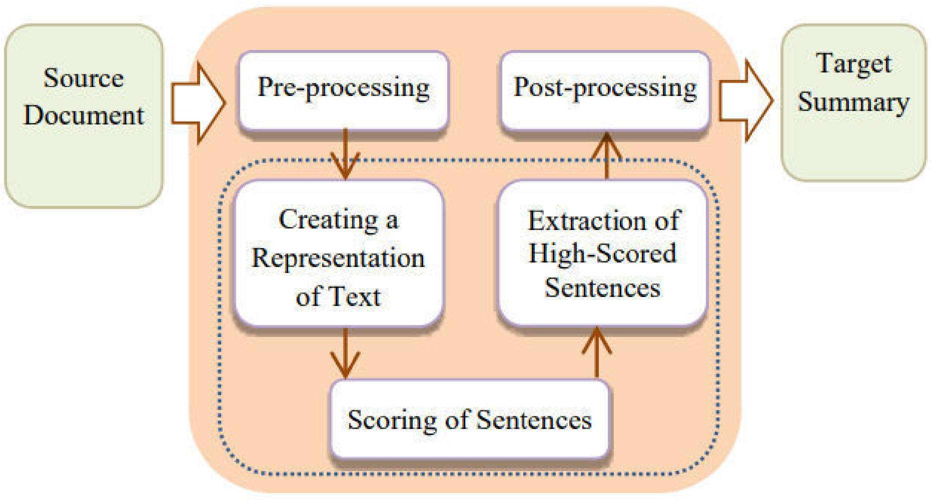


Figure 16: The architecture of an extractive text summarization process (Source (El-Kassas et al., 2021))

4.2 Abstractive text summarization

In comparison to extractive summarization, abstractive summarization does not copy sentences (or subsequncies of tokens) from the source text to generate the summary. It requires the ability to generate new sentences (Bhat et al., 2018). Thus, this relates to natural language generation (*NLG*) task, in which new phrases or words are generated.

The architecture of an abstractive text summarizer is depicted in Figure 17. It comprises of pre-processing and post-processing, as well as processing jobs, which include the following:

1. constructing an internal semantic representation, which depends on specific *NLG* model;
2. creating a summary utilizing *NLG* techniques in order to produce a summary that is fluent in terms of sentences connection (absolutely different to extractive summarization).

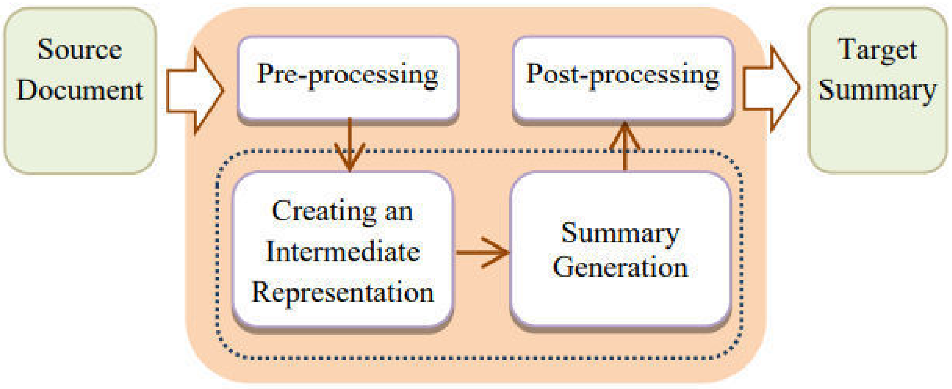


Figure 17: The architecture of an abstractive text summarization process (Source (El-Kassas et al., 2021))

4.3 Hybrid text summarization

Automatic hybrid text summarization is a combination of abstractive and extractive summarization. As illustrated in Figure 18, this kind of summarization begin by pre-processing the text that will be sent into the extractive summarizer. This summarizer removes a few sentences or phrases from the original text and leaves the remainder in its original order but without the deleted token sequences. We next move to *NLG* by creating abstractive summaries from the output of the extractive summary in order to receive the output of *NLG* model. If necessary, this output may be further post-processed.

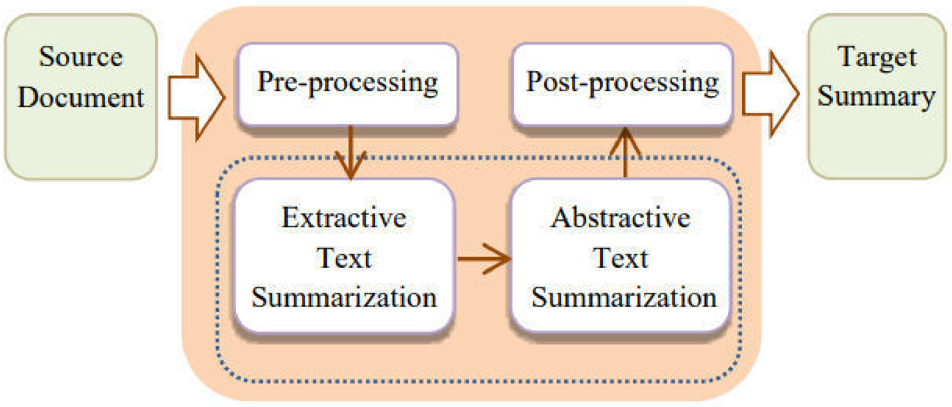


Figure 18: The architecture of an hybrid text summarization process (Source (El-Kassas et al., 2021))

4.4 Unified text to text transformer (T5)

Choosing an abstract summarizing model is not straightforward, as some *NLG* models, such as GPT3, produce unmatched state-of-the-art performance in summarization yet require a vast number of parameters (Floridi et al., 2020). Most importantly, GPT-3 is an API-based model, which means that there would be no our experimentation where we would call the API and OpenAI[1] would train the entire model instead of us.

As a result, we researched models with fewer neural network weights, such as PEGASUS, BART, and the UNIFIED TEXT TO TEXT TRANSFORMER (T5). Comparisons of these models have already been conducted on the summarization problem using BBS News data, where T5 clearly outperform the others (Gupta et al., 2021). Another intriguing study on the abstract summary is (Garg et al., 2020) by comparing BART and T5 and using a different News dataset, where T5 again outperforming BART. As a result, we choose to focus our research on *abstract summarization* utilizing the T5 transformer.

In article (Raffel et al., 2019), the T5 model was introduced. This article offers a preliminary examination of the application of contemporary techniques to natural language processing. To evaluate various approaches, different models were trained on a variety of different datasets and using a variety of different training strategies. Additionally, the authors presented a model T5 as a framework that aims to integrate a number of downstream tasks in natural language processing and *NLG* into text-to-text transformation.

T5 has **encoder-decoder** architecture as illustrate in Figure 19. The decoder, like the encoder (see Encoder in section 2.5.3), is constructed of many layers. Additionally, the decoder comprises two levels of multi-head self-attention, whereas the encoder only contains one. The encoder's output is used as the input to this new sublayer. In comparison to the encoder, the decoder is *unidirectional*. T5 transformer

[1]https://openai.com/

has exactly similar encoder to BERT which transform text into a vector, but additionally the decoder convert this vector into text.

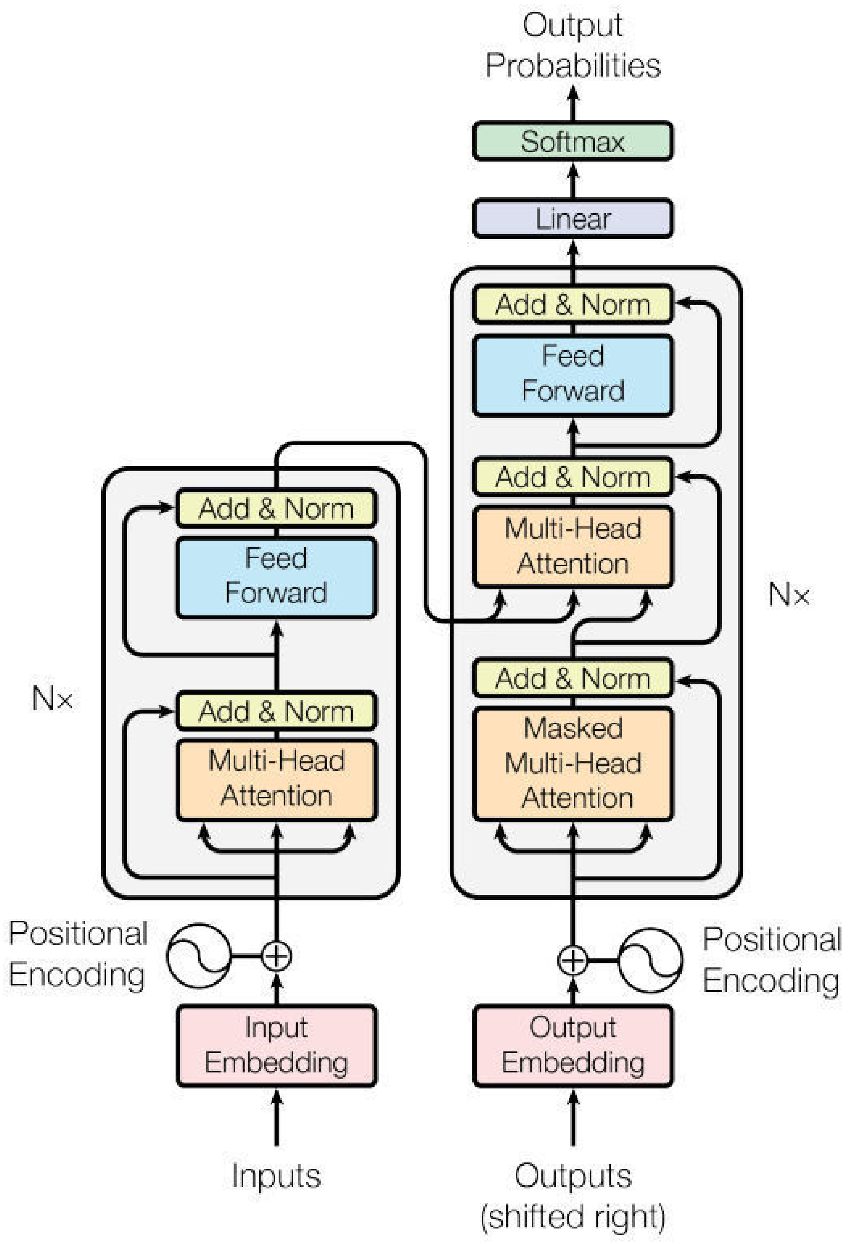


Figure 19: T5 attention mechanism (Source (Vaswani et al., 2017))

4.4.1 T5 framework

The T5 model, which has a partially comparable architecture to the BERT model, is trained using the processes described in Subection 2.5.3, namely word prediction based on mask context and the next sentence's relevance. Following that, these pre-trained models are fine-tuned to do specific tasks, such as predicting the two labels for sentiment analysis like text classification or generate summary from given text. On the other hand, the T5 framework proposes to use the same model, loss function, and hyperparameters for multiple natural language processing and *NLG* downstream task.

For this model, the input data must be structured in such a way that the model can determine which task should be used. The answer is then a straightforward textual form, where a specified prefix indicating which task T5 should focus on. The T5 model's fundamental inputs are illustrated in Figure 20. On the left side of this Figure, we must provide an input text with a specific prefix that instructs the form of output text.

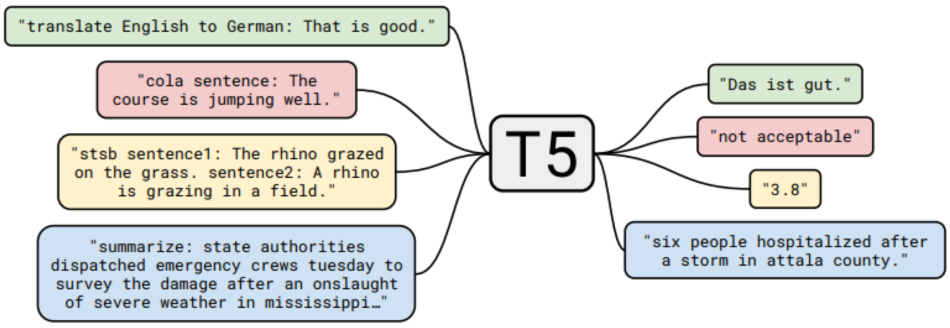

"translate English to German: That is good."
"cola sentence: The course is jumping well."
"stsb sentence1: The rhino grazed on the grass. sentence2: A rhino is grazing in a field."
"summarize: state authorities dispatched emergency crews tuesday to survey the damage after an onslaught of severe weather in mississippi…"
T5
"Das ist gut."
"not acceptable"
"3.8"
"six people hospitalized after a storm in attala county."

Figure 20: T5 input alternatives with prefix (Source (Raffel et al., 2019))

4.4.2 Selection of data for pre-training

The most significant advantage of the study (Raffel et al., 2019) is that this model was trained using only pure data. It has been demonstrated that only high-quality clean data can be used to train the T5 transformer and show accurate results. Just like the technique of model training from scrach, the dataset used to predict the model is an integral part of the transfer learning approach. A stereotype in the development of such models is the use of extensive unlabelled training corpora, i.e. corpora without information from the learner. Among such datasets include the Common Crawl corpus containing petabytes of text from all different web pages over the last 12 years, i.e. approximately 20TB of data each month. Data is obtained primarily from HTML files. However, such a dataset contains a large number of redundant texts - such as duplicated texts, error messages and others. The authors of the paper have examined this corpus in several ways and created the C4 corpus (Colossal Clean Crawled Corpus) such that (Raffel et al., 2019):

1. they deleted any page with fewer than five phrases and kept only lines with at least three words;
2. they kept only lines that ended in a punctuation mark (i.e. a period, exclamation mark, question mark, or end quotation mark);
3. because several of the scraped pages had warnings about the need to enable Javascript, they deleted any lines containing the token Javascript;

4. they took down any page that featured a word from the List of „Dirty“, „Naughty“, „Obscene“, or other „Bad Words“;
5. a few pages contained code accidently. Because the curly bracket „{“ is used in a variety of computer languages (including Javascript, which is extensively used on the web), but not in natural text, they eliminated any pages that utilized it;
6. some sites contained placeholder text for „lorem ipsum“, they eliminated any page that contained the two gram „lorem ipsum“
7. to deduplicate the data set, they removed all but one three-sentence span that appeared in the data set more than once.

4.4.3 T5 fine-tuning

As mentioned previously, we utilize the T5 transformer as a „black model“, which means we just feed a text with a particular prefix and the model generates the required result. To fine-tune the model for the summarization task, we append the prefix „summary:“ to the text that we wish to summarize, along with gold summmary. This essentially means that we must input pairs comprising text for summarizing and expected summaries and then fine-tune the model on top of it.

There are two alternate techniques to fine-tuning that affect only a portion of the parameters (or weights) (Raffel et al., 2019).

The first approach, dubbed „adapter layers“ (Houlsby et al., 2019; Bapna et al., 2019), is motivated by the need to maintain the majority of the original model while fine-tuning. Adapter layers are additional dense-ReLU-dense blocks that are added after each of the Transformer's pre-existing feed-forward networks. These innovative feed-forward networks are constructed in such a way that their output dimensions match their input dimensions. This enables them to be integrated into the network without requiring any additional structure or parameter adjustments. Only the adapter layer and layer normalization settings are updated during fine-tuning. The primary hyperparameter in this strategy is the feed-forward network's inner dimension d, which affects the number of new parameters introduced to the model.

The second approach of fine-tuning is called „gradual unfreezing“ (Howard et al., 2018). Gradually unfreezing the model allows for fine-tuning of an increasing number of its parameters over time. Gradual unfreezing was initially used to a single-layer language model architecture. In this setting, only the parameters of the final layer are changed at the start of fine-tuning, followed by the parameters of the second-to-last layer after a certain number of updates, and so on until the complete network's parameters are fine-tuned. To adapt this strategy to our encoder-decoder T5 model, we gradually unfreeze layers in both the encoder and decoder, starting at the top in both cases.

5. Proposed summarization method

In our work we will show that a suitable fine-tuning of the T5 transformer may lead to the excellent results. Moreover, we have proposed a method to improve T5 summarization even more. According to us, the main essence of the great results of T5 transformer is not the possibility to apply it on multiple tasks, but the training of this model on clean and high quality data (see Subsection 4.4.2). That is why we have created a pre-process the input before we fine-tune the T5 transformer, so that the input contains the most relevant sentences of the certain text from which we want to create a summary. Thus, we first reduce the number of sentences of the underlying text and then create an *abstractive summary* on top of this *extractive summary*.

Let us first describe the process of this hybrid summarization as a whole, shown in Figure 21. Before that, just recall that *ruling comments* are all comments of the fact-checker related to the veracity of a certain claim and *justification* is a gold summary of these ruling comments manually created by the fact-checker (see for example figure 1). If we start describing this new procedure from the left side to the right side of Figure 21, we take ruling comments and justification as a pair which is related to the given claim. And we use this pair as input to the T5 transformer by successive modifications.

Justification itself is not subject to heavy editing because it is a golden summary manually created by a human. An example of a modification of this is, for example, the removal of URLs or the reduction of multiple blanks to a single blank in the text.

The contribution of this work is tied to a sophisticated modification of sentences from ruling comments of some claim. Thus, we will focus on the upper part of Figure 21, where these ruling comments are first pre-processed, as with justification, but then the text of it is split into individual sentences. Right after, we explore 3 different approaches to creating embeddings of all these sentences.

The first approach to creating embeddings is the one we have specifically trained, via SENTENCE-BERT. We propose a special way to create the data to fine-tune SENTENCE-BERT. We will explain the specific algorithm below in section 5.1. For the moment, we can understand it as construction of triplets of sentences located in ruling comments. We apply these triplets to fine-tune SENTENCE-BERT in order to improve its vectorization quality. Thus, after extracting these triplets, we will retrain a new SENTENCE-BERT model to vectorize each single sentence of the ruling comments.

The two others consist in creating sentence embeddings via DOC2VEC or TF-IDF, respectively, where in both cases, we convert the words of all sentences to lemmas. In the case of DOC2VEC, we feed all sentences into the DOC2VEC model at once. DOC2VEC is trained only on these sentences but vectorizes each sentence separately. In the case of TF-IDF, after lemmatization is done, we create a vector whose i-th element represents weight of i-th token in vocabulary of rulings comments for the claim.

As we have described, the whole process of hybrid summarization is distinguished by several kinds of sentence vectorizations. Thus, as soon as we vectorize the sentences (by any method), we compute the LOF for each sentence by means of the sentence vectors. In this case, LOF can be understood as a kind of ranking, which ranks sentences according to their importance in the text. Then we remove some sentences, namely, the ones that are the most different. We empirically derived the ideal percentage of sentences to be removed, to be explained later. Now, let us further assume that we have removed some sentences. We combine the individual sentences that have not been removed in the original order to form a continuous text containing fewer sentences than the original, which is actually an *extractive summary*.

Next, our pre-processed justification will form a new pair with the extractive summary. So far we have explained only one pair associated with one claim for the explanation. In reality, we split the whole data into many training pairs, validation pairs and test pairs (we will explain the specific split ratio in the next chapter 6). Let us focus on the first frame from the right in Figure 21 named „T5 abstractive summarization“. For fine-tuning, we apply only the training and validation pairs. The training pairs enter the model sequentially in the form of several pairs as *batches*. Validation pairs are only applied to test and optimize parameters on data that are not applied during fine-tuning. Next, focusing on this box in Figure 21, we generate automatic summaries using only the extractive summaries from the test pairs (the pairs that were not used in the fine-tuning). Finally, we compare these automatically generated abstractive summaries with the golden justifications from the test pairs using the Rouge metric, which evaluates the quality of the summary.

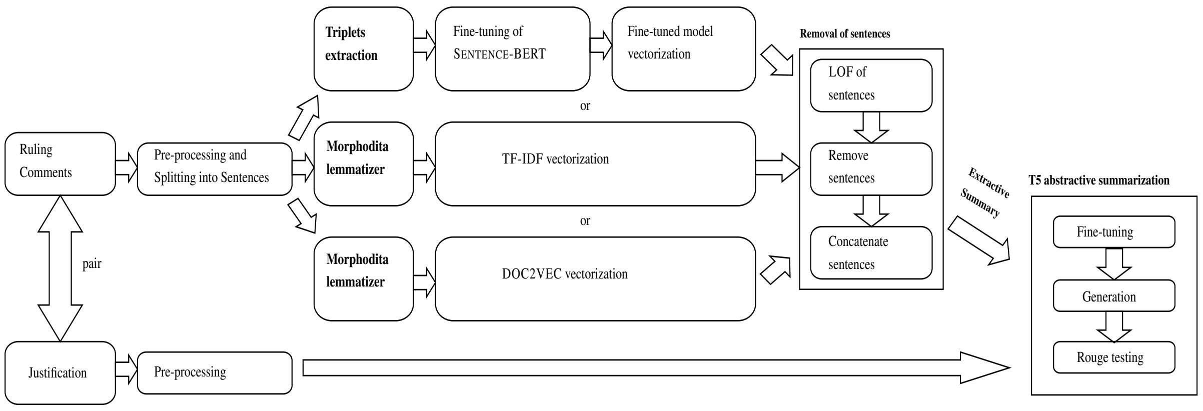


Figure 21: Proposed hybrid summarization process

5.1 Detailed extraction of triplets

At the beginning of this chapter, we have only mentioned triplets very gently, so that the whole process of the hybrid model would not be even more complicated. However, it is important to pay attention to the fact that the creation of triplets influences the quality of vectors after SENTENCE-BERT has been fine-tuned. As we mentioned in section 2.5.4, one of the possibilities of fine-tuning SENTENCE-BERT is to minimize the *triplet loss objective function* (59). Assuming we have one triplet, by fine-tuning on it we tell SENTENCE-BERT to retrain itself to minimize the distance between two similar sentences, and, vice versa, to minimize the distance between two dissimilar sentences. For this we only need three sentences, where one is the *anchor*, i.e. the sentence with respect to which we will minimize the distance of another sentence and maximize the distance with yet another sentence. The sentence with respect to which the *anchor* will minimize the distance is called *positive*. Conversely, *anchor* will maximize the distance with respect to a sentence we call *negative*.

The algorithm for triplet extraction, which helped us to fine-tune the SENTENCE-BERT, is properly described in Algorithm 1. For simplicity, let's denote **ruling comments** as **report** and manual **justification** as **summary**.

Let us now describe Algorithm 1 line by line.

- In the first line we define the inputs, where RC is the set of reports, k is the number of these reports, and SM is the corresponding manual summary of these reports, so it is clear that their number must be equal to k. The last input is an integer t, which we use as a constant in this algorithm.
- The second line defines the output:
- The third line indicates that through each single report and its corresponding summary we perform some actions. Lines 4 to 10 declare the variables.
- In the eleventh line, we assign a summary vector from the i-th summary to $SMVEC$ (summary vector), using the general and not fine-tuned SENTENCE-BERT.
- In line 12, for the i-th Report, we split its sentences and form them into the $RCSPLIT$ set. In the next line, we just count the number of these beliefs as the value of n.
- In lines 14–17 we create an embedding for each sentence of the report by applying the same SENTENCE-BERT model as in line 11, and compare the distance of each sentence from the summary using the *cosine distance*.
- In line 18 we sort the sentences of the $RCSPLIT$ set according to their *cosine distance* to the summary in descending order, and store this ordered set in the $RCSPLIT_{desc}$ variable.
- In line 19, from these sorted sentences we select the t sentences that are most similar to the summary, and assign them to the POS set, and in the next line we select the t sentences that are least similar to the summary and store them in the $NEGATIVE$ set. Thus, we have created two distinct sets that contain at most t text sentences from the i-th report.
- Now we simply need, for each $anchor$ sentence in the POS set, to randomly select another sentence from the POS set as $positive$ and randomly select another sentence $negative$ from the $NEGATIVE$ set. The $anchor$ cannot be identical to the $positive$ sentence because their distance could no longer be minimized. Thus, in line 21, for each anchor in the set of positive sentences POS, we will create a set in which the anchor will not be present, denoted as $POSITIVE$ (see line 25). And in lines 26 and 27 we will randomly select first the $positive$ sentence and then the $negative$ sentence to the $anchor$.
- Finally, in line 28 for each anchor and each report we will store the triplet in the $TRIPLETS$ set; we thus get a bunch of triplets. When this algorithm is finished, we should get a total of $t \cdot k$

triplets.

```
Input: $RC \leftarrow \{RC[1],\ldots,RC[k]\},\ SM \leftarrow \{SM[1],\ldots,RC[k]\},\ t \in \mathbb{Z}$
Output: $TRIPLETS \leftarrow \{TRIPLETS[1],\ldots,TRIPLETS[t \cdot k]\}$
for $i \leftarrow 1$ to $k$ do
    $SMVEC \leftarrow \emptyset$
    $RCSPLIT \leftarrow \emptyset$
    $SVEC \leftarrow \emptyset$
    $POS \leftarrow \emptyset$
    $NEGATIVE \leftarrow \emptyset$
    $COS \leftarrow \emptyset$
    $n \leftarrow 0$
    $SMVEC \leftarrow$ sbert$(SM[i])$ // vectorize $i$-th summary
    $RCSPLIT \leftarrow$ split$(RC[i])$ // split $i$-th report into sentences
    $n \leftarrow |RCSPLIT|$
    for $j \leftarrow 1$ to $n$ do
        $SVEC[j] \leftarrow$ sbert$(RCSPLIT[j])$// vectorize j-th sentence in $i$-th
            report
        $COS[j] \leftarrow$ d$(SVEC[j], SMVEC)$ // calculate cosine distance
            between j-th sentence in $i$-th report and $i$-th summary
    end
    $RCSPLIT_{desc} \leftarrow$ orderdesc$(RCSPLIT, COS)$// order sentences in $i$-th
        report by cosine similarity
    $POS \leftarrow$ top$(RCSPLIT_{desc}, t)$ // t most similar sentences to summary
    $NEGATIVE \leftarrow$ tail$(RCSPLIT_{desc}, t)$ // t least similar sentences to
        summary
    foreach $anchor \in POS$ do
        $POSITIVE \leftarrow \emptyset$
        $positive \leftarrow \emptyset$
        $negative \leftarrow \emptyset$
        $POSITIVE \leftarrow POS \setminus anchor$ // remove anchor from positives
        $positive \leftarrow$ random$(POSITIVE)$
        $negative \leftarrow$ random$(NEGATIVE)$
        $TRIPLETS \leftarrow TRIPLETS \cup (anchor, positive, negative)$ // save each
            new triplet
    end
end
```

Algorithm 1: Triplets extraction

5.2 Detailed extractive summarization

To our knowledge, no project has so far attempted to extract summaries from fact-checking reports or other data via the *Local Outlier Factor* as a pre-process for any transformer. Actually, by applying such an *extractive summarization*, we can improve the subsequent abstractive summarization, which can be classified as a *NLG task*. Therefore, in this section we consider it necessary to highlight in detail the algorithm (see Algorithm 2) of this extractive summarization, which is the main contribution of this work.

Again, for simplicity, let us denote the **ruling comments** for a certain claim as a **report**. The whole algorithm is described in Algorithm 2.

- The input to this algorithm is the set k of RC reports and a parameter p, which denotes the percentage of sentences we want to delete from all reports RC. Note that this is not the percentage of sentences that we want to remove from each report.
- In our work, of course, we do not want to remove all sentences from a report, so we necessarily have to require k final extractive summaries in the second line as an output. In lines 3 to 11 we declare the sets, but in this case we only declare them once compared to the first algorithm 1 where we declared the variables for each single report.
- In line 12 we go over each report index in the RC set again. According to line 14, we first split the report into sentences and store their number in the i-th element of the $RCSPLIT$ set. We assign the number of sentences in this report to the variable n in line 15. The splitting of the sentences preserves their order, so for each sentence of the report we store its order in the $SORDER$ set, where $RCSPLIT[i][j]$ denotes the order of the j-th sentence in the i-th report (line 17).
- After storing the sentence order, we convert each sentence into a vector. The vectorize function (line 18) takes the sentence text as input and outputs a vector via TF-IDF, DOC2VEC or SENTENCE-BERT, respectively. In the case of TF-IDF, we assume that the inverse document frequency for each word in the report has been already calculated in advance. In the case of DOC2VEC, we similarly assume that each token has a certain vector. Finally, for SENTENCE-BERT we assume that the optimization via triplets in the algorithm 1 is complete and the model is already fine-tuned.
- Thus, we now have a vector for each sentence in the report, where we store the vector of the j-th sentence and the i-th report as $SVEC[i][j]$ (see line 18).
- Next, we compute the LOF value for each vector using the j-th vector in the i-th report $SVEC[i][j]$ and the set of all vectors for the i-th report $SVEC[i]$ (shown in lines $20 - 22$).
- Now imagine that we have been given the pairs forming a sentence and its LOF within the report to which it belongs. Thus, we are no longer interested in the sentence vectors, but rather in the LOFs of the sentences that we have computed from the vectors. In line 24 we sort all the sentences of the whole dataset according to their LOF in order to find the sentences with the largest LOF in the whole dataset. By sorting the sentences in descending order in line 24, we just cut off a certain rounded percentage p from the top of this sort by using the function „deletepercentage“ function

in line 25.

- Since we have already saved the order of the sentences in the $SORDER$ set, we will sort the sentences of each report in lines $26 - 28$ in the order of the successive sentences, although some indexes of the order will be missing, because some sentences have been deleted.
- In line 27 we concatenate these sentences to create a fluent text with fewer sentences than the original one.
- Finally, in lines $30 - 34$ we check if some whole reports have been deleted, which would lead us to reduce the percentage of deleted sentences because removing all sentences of report does not make sense. We have tested different values of value p and it obviously depends on the dataset type.

```
[1] **Input:** $RC \leftarrow \{RC[1], \ldots, RC[k]\},\ p \in (0,\ 1)$
[2] **Output:** $ERC \leftarrow \{ERC[1], \ldots, ERC[k]\}$
[3] $RCSPLIT \leftarrow \emptyset$
[4] $RC_{concatenated} \leftarrow \emptyset$
[5] $RCSPLIT_{LOFdesc} \leftarrow \emptyset$
[6] $RCSPLIT_{SORDERdesc} \leftarrow \emptyset$
[7] $RCSPLIT_{extracted} \leftarrow \emptyset$
[8] $SORDER \leftarrow \emptyset$
[9] $SVEC \leftarrow \emptyset$
[10] $SVEC_{desc} \leftarrow \emptyset$
[11] $LOF \leftarrow \emptyset$
[12] for $i \leftarrow 1$ to $k$ do
[13]     $n \leftarrow 0$
[14]     $RCSPLIT[i] \leftarrow$ split($RC[i]$) // split $i$-th report into sentences
[15]     $n \leftarrow |RCSPLIT[i]|$ // get number of sentences
[16]     for $j \leftarrow 1$ to $n$ do
[17]         $SORDER[i][j] \leftarrow j$
[18]         $SVEC[i][j] \leftarrow$ vectorize($RCSPLIT[i][j]$) // vectorize j-th sentence
[19]     end
[20]     for $j \leftarrow 1$ to $n$ do
[21]         $LOF[i][j] \leftarrow$ LOF$_{Norm}$($SVEC[i], SVEC[i][j]$) // calculate $LOF_{Norm}$ for
                j-th sentence
[22]     end
[23] end
[24] $RCSPLIT_{LOFdesc} \leftarrow$ orderalldesc($RCSPLIT$, $LOF$) // order all sentences in
        dataset by their $LOF_{Norm}$
[25] $RCSPLIT_{extracted} \leftarrow$ deletepercentage($RCSPLIT_{LOFdesc}$, $p$) // delete p
        percentage of sentences from dataset
[26] for $i \leftarrow 1$ to $k$ do
[27]     $RCSPLIT[i]_{SORDERdesc} \leftarrow$ orderasc($RCSPLIT[i]_{extracted}$, $SORDER[i]$) // sort
            the sentences according to their ordering in the initial
            report
[28]     $RC[i]_{concatenated} \leftarrow$ concat($RCSPLIT[i]_{SORDERdesc}$) // create plain text
            from sentences
[29] end
[30] if $|RC_{concatenated}| \neq k$ then
[31]     break
[32] else
[33]     $ERC \leftarrow RC_{concatenated}$ // save extractive summary of $i$-th report
[34] end
```

Algorithm 2: Extractive summarization using LOF$_{Norm}$

6. Experiments

In this chapter we first show how our proposed fine-tuned SENTENCE-BERT identifies significant outliers as report sentences which will be removed. After some sentences are removed and the remaining sentences concatenated by initial ordering within reports, T5 transformer is fine-tuned to generate an *abstractive* summary. Note that abstractive summarization belongs to the *NLG* task. Thus, when generating a summary applying the T5 transformer, new sentences can be created, which are often paraphrases of the correct summary. On the other hand, each sentence forming the *extractive* summary must be included in the source text from which the summary is taken out, because we extract a subset of sentences from the set of sentences involved in the source text.

We will compare this fine-tuned SENTENCE-BERT on fact-checking datasets in Czech and English. Furthermore, we will compare the fine-tuned SENTENCE-BERT with other kinds of contextual text representations such as TF-IDF and DOC2VEC as well. Additionally, proposed extractive method, that serves as a pre-process for abstractive summarization by the T5 transformer, on the largest Czech news dataset. We actually want to see if we are able to improve the ability to generate a summary with not-fine-tuned SENTENCE-BERT and even on a completely different type of data, compared to fact-checking data.

6.1 Hyperparameters

The results on each data set we used will differ not only by the diverse nature of the data but also by the **hyper-parameters** that we specify when fine-tuning the final T5 or MT5 transformer. The difference in **hyperparameters** and **parameters** is that we understand parameters as weights of the neural network or weights of the sentence vectors, so they are the *result* of neural network training (or fine-tuning). In contrast, *hyperparameters* are necessarily parameters that *precede* the model training (or fine-tuning).

In the following text, we will consider certain *hyperparameters* that have been experimentally tested to achieve the best results. The following *hyperparameters* were taken into account:

1. **batch size** - the number of pairs (text and its summary) that enter the T5 model at a time during fine-tuning,
2. **number of epochs** - number of cycles of fine-tuning iteration (we train all T5 models on 3 epochs),
3. **max words text** - the maximum number of words for the text to be summarized (the text longer than this hyper-parameter will be truncated to a length equal to the hyperparameter),
4. **max words summary** - the maximum number of words of the summary (the text longer than this hyper-parameter will be truncated to a length equal to the hyperparameter),
5. **parameter** p (see Algorithm 2) - determines the percentage of sentences to be removed from the

entire data set during the extractive summarization,

6. **number of neighbours in LOF** - we set this parameter to a constant value of **4** for all datasets; when trying other values, this value clearly dominated them.

6.2 Evaluation metrics

In this research we experimented with datasets in English and Czech. Our T5 transformer is included in the models for *NL*. For evaluating these models we use the **ROUGE metric** (Lin, 2004), which has been previously utilized as the sole or primary metric in all studies cited in our work that discuss automatic summarization (see for example (Gehrmann et al., 2021), (Atanasova et al., 2020) or (Kazemi et al., 2021)).

The *Rouge* measure was originally developed to assess the quality of natural language generation in English, and it is applied to the English dataset in this study. It makes use of an English stemmer, stop words, and synonyms. The original *ROUGE* metric evaluates the generated summary's quality automatically by comparing it to a reference summary created by humans. There are two distinct variations. *ROUGE-N* quantifies the amount of overlap between the generated and reference summary in terms of N-grams. *ROUGE-L* examines the reference and the generated summaries for the longest common subsequence. **ROUGE-1**, **ROUGE-2** and **ROUGE-L** were used to evaluate our methods for English datasets.

Currently, the only metric for evaluating automated summarization in Czech is **ROUGERAW**, which we use for Czech datasets. Straka et al. in (Straka et al., 2018) proposed *ROUGERAW* as a language-independent variation of ROuGE. *ROUGERAW* is language-independent because it does not require stemmers, stop words, or synonyms. It is used to determine the recall, precision, and F-score. Additionally, it also features two versions, *ROUGERAW-N* and *ROUGERAW-L*, which match the original ROUGE metric's variants. **ROUGERAW-1**, **ROUGERAW-2**, and **ROUGERAW-L** were chosen to evaluate our approach on Czech datasets.

6.3 Experimental framework

The whole experimental part was programmed in the **Python programming language**. In our publicly available github[1] repository we store the whole code applied in this work. Unfortunately, due to memory requirements (the models and data had several gigabytes) we processed the work on the university cloud, which was unable to share the data for public use. On the other hand, the scripts in this repository can be run sequentially from scraping the whole data to generating or fine-tuning the T5 transformer. For T5 transformer fine-tuning we used **GPU** server graphics cards, namely **Nvidia A100** and **Nvidia A40**.

[1]https://github.com/petervajdecka02947/MasterThesis2022

Please, note that running the whole process from scraping through text generation to the evaluation of the model can take from several up to tens of hours.

For English, we have fine-tuned the **T5 Base** model (Raffel et al., 2019) with **220M** parameters or weights of its neural network. For the Czech language, a multilingual model pre-trained for **101** languages was used. This model is named **mT5 Base** (Xue et al., 2020) transformer model and has **580M** parameters of its neural network.

6.4 Datasets

6.4.1 Politifact dataset

The reason why we have chosen Politifact as the dataset we will be using is that the Politifact site contains the summary of ruling comments for a huge number of claims. Furthermore, automatic summaries have already been created on the data from this page by other researchers (Atanasova et al., 2020; Kazemi et al., 2021), which of course lead to an attempt to improve these results. The first summary dataset was created as described by (W. Y. Wang, 2017), but unfortunately it does not contain justifications for the claims, and, subsequently another dataset (Alhindi et al., 2018) was created, which contains justifications but does not contain the ruling comments from which the justifications have been created. As regards the research by (Atanasova et al., 2020) and (Kazemi et al., 2021), the data used by the authors are not publicly available. This led us to scraping the data directly from the Politifact site, currently as of **17-Feb-2022**.

Scraping data from Politifact

As we have mentioned several times in this paper, we see the main benefit of the work done by (Raffel et al., 2019) to be simple but extensible processing of crawled data leading to cleaner and better-quality data. The authors have actually created a new, clean, C4 dataset, and have trained the T5 transformer on it. Since we decided to use the Politifact data to fine-tune the T5 transformer, which is also from the web as the original C4 training data, we also decided to select cleaner and higher-quality data from Politifact.

When we look at the creation of the raw data in paper by (Alhindi et al., 2018), the authors remove all sentences from the justification that contain words related to the plausibility of the claim. We have removed the words in the same way; these words can be found in github[2].

The other main modification applied in article (Alhindi et al., 2018) was to approximate justifications, in cases when they did not exist, as the **last 4 sentences** of the ruling comments. We did not subscribe to this practice, because it is then impossible to reliably determine whether this set of 4 sentences truly

[2]https://github.com/Tariq60/LIAR-PLUS/blob/master/forbidden_words.txt

expresses a summary.

We have therefore made the following modifications to the creation of our new dataset:

1. only pages containing "Our ruling" or "Our Rating" were considered, ensuring that the justification had been manually created,
2. then characters like „\n“ or „\t“ were removed to assure fluency of the text,
3. we removed the HTML tags and URLs,
4. in the justifications, sentences containing a word relevant to the veracity was removed, as mentioned above,
5. we replaced every consecutive 2 or more spaces with a single space,
6. we updated all data from past until 17 February 2022.

Compared to the research by (Alhindi et al., 2018), where the number of records is 12,836 in the whole data, in our data we have **12891** records. As we can see in Table 5, the average number of tokens in the ruling comments is 793 words, and there are 85 words in the justification on average. The minimum number of words of ruling comments for a single report is 40 words, and for the justification it is only 2 words (this may be the result of removing sentences expressing the veracity). The quartiles and maximum values of ruling comments are needed when specifying the hyperparameters of the model.

We divided these data in the same proportion as in the above papers, i.e. 80 percent as the training part (10312 records), and 10 percent as the validation part (1289) and testing part (1290 records).

	Ruling comments length	**Justification length**
count	12891.00	12891.00
mean	793.71	85.51
std	289.86	42.38
min	40.00	2.00
25%	589.00	58.00
50%	755.00	80.00
75%	953.00	106.00
max	2935.00	1121.00

Table 5: Descriptive statistics of tokens on the Politifact dataset

6.4.2 Demagog dataset

The fact-checking dataset in the Czech language that we dealt with was Demagog (Přibáň et al., 2019). These data are very similar to Politifact, but the summaries of the reports themselves are not as developed and not as numerous as in the case of Politifact. In the Czech-language NLP research we have not yet seen any work that would address the summarization of Czech fact-checking reports, which of course entailed a similar problem in obtaining data as with the Politifact. The reason for this was that there is no publicly available data or even query interface to export data that would contain ruling comments from Demagog in conjunction with their justifications.

Therefore, we had to get acquainted with the Demagog **API**[3] and create the unique query for the **GraphQL** database. This query is shown in Listing 6.1 along with an end-point for sending a POST request to extract the data in the form we request. We will not describe this query in detail, but it may help in extracting the data we used in this work.

```

graphql_query = '''
{
    statements(
    limit: 13000,
    sortSourcesInReverseChronologicalOrder:true){
    id
    excerptedAt
    source{releasedAt}
    title
    content
    sourceSpeaker{
        id
        firstName
        lastName
        role
   }
    assessment{
      explanation
      shortExplanation
      shortExplanationCharactersLength
      veracity{
        name
      }
}
}
}
'''
end_point = 'https://demagog.cz/graphql'
```

Listing 6.1: GraphGL query to export data

The Demagog data has a total of 3406 entries (see Table 6); while the average number of words in the ruling comments for an average claim is 295, the minimum number is 32 words and the maximum

[3]https://demagog.cz/stranka/api-pro-vyvojare

number is 1531 words. In the case of justification, we have an average word count of 30, where the minimum word count is 4 and the maximum word count is 95. We divided this data in the same proportion as shown in the papers above, i.e. 80 percent as the training part (2724 records), and 10 percent as the validation part (341) and testing part (341 records).

	Ruling comments length	**Justification length**
count	3406.00	3406.00
mean	295.45	30.26
std	179.22	9.92
min	32.00	4.00
25%	169.00	23.00
50%	253.00	31.00
75%	380.00	38.00
max	1531.00	95.00

Table 6: Descriptive statistics of tokens on the Demagog dataset

6.4.3 SumeCzech dataset

SumeCzech is the largest Czech data corpus of news (Straka et al., 2018). As we already mentioned, the fact-checking datasets are relatively small. We however wanted to test how our model would perform on the big summary data of same nature. And next we wanted to know, if we select only 10 percent of the training data of this dataset, whether the pre-trained model can be compared with other state-of-the-art models trained on the whole data from scratch. Thus, we will focus on comparing fine-tuning on 10 percent of data versus training from scratch on the whole data.

The data also includes an „out of domain“ data set that contains test data from sources other than the training data and validation data. Since amount of data is really huge, we consider this dataset as a supplement, where we just applied not-fine-tuned SENTENCE-BERT, i.e., only the algorithm of the extractive summarization 2; in contrast, the T5 transformer is always being fine-tuned. The data can be exported via a Python script from the SumeCzech website[4].

We will generate a **headline** from the **news text**. As we can see in Table 7, the total number of entries in the data is approximately 211 thousand, the average text length is 401 words, and the average number of words for headlines is almost 9. The minimum word count is 100 for texts and 3 for headlines. The maximum number of words in the text of the news is more than 13 thousand, and in the headlines it is only 22 words. We obtain data consisting of 77866 cases in the training set, 44567 in the development set, 44454 in the test set and 44976 in the Out of Domain Test set.

[4]https://ufal.mff.cuni.cz/sumeczech

	Text length	**Headline length**
count	211863.00	211863.00
mean	401.38	8.76
std	307.25	2.45
min	99.00	3.00
25%	224.00	7.00
50%	319.00	9.00
75%	473.00	11.00
max	13283.00	22.00

Table 7: Descriptive statistics of tokens on the SumeCzech dataset

6.5 Testing part of datasets

In this section, we first focus on the evaluation results within each dataset and then on the different forms of token count distribution in the test part of the datasets. Thus, we will display four different graphs in columns. The first two graphs will contain the token difference on the test set for the ruling comments. In one of the graphs we will show the distribution of tokens before the proposed extractive summarization and in the other the distribution of tokens after the extractive aggregate. Another pair of graphs will probably be more interesting which is the comparison of the token distribution of our generated summary with the token distribution of the golden justifications.

6.5.1 Politifact results

The test set contained a total of 1290 pairs in the form of rulings comments and the corresponding justification. The specific results are shown in Table 8. We can see that our proposed method achieved, objectively, the best results. We can also see that applying the hybrid model with extractive summarization via LOF improved in every case the results of the *Baseline* T5. This baseline model was only fine-tuned without any extractive summarization. Furthermore, it was shown that the presence of a claim by concatenation with the extractive summary certainly improved the results of the overall automatic summary.

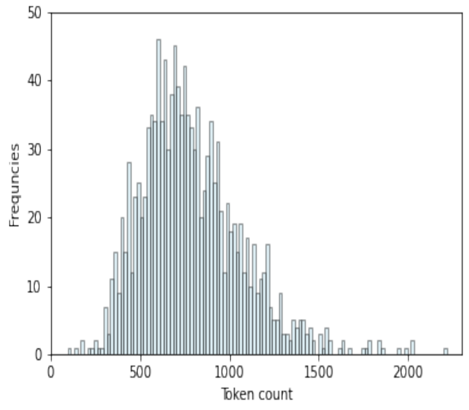
Since we have reduced the number of words by means of extractive summarization, we also looked at how the distribution of words changes as we reduce the number of sentences in the best model from Table 8. Thus, in this case, it is a reduction of 13 percent of the total data in terms of sentences in testing data.

In Figure 22, we visualized the distribution of the number of ruling comment tokens before and after the extractive summarization was done. We have omitted only one report that is over 2300 tokens long for better visualization. On the x-axis we define a certain number of tokens in the ruling comments of a

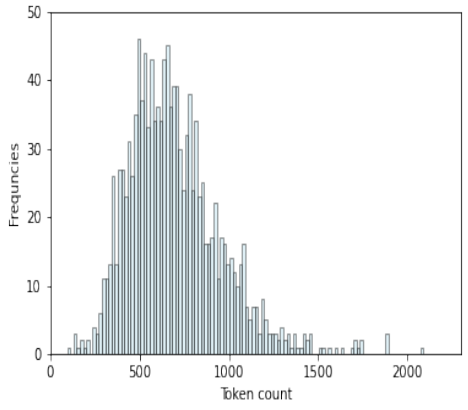
Politifact.com				
Source	**System**	**Rouge 1**	**Rouge 2**	**Rouge L**
Atanasova 2020 (University of Copenhagen)	Explain-Extractive	35.7	13.51	31.58
	Explain-MT	35.13	12.9	30.93
Kazemi 2021 (University of Michigan)	TextRank	27.74	7.42	23.24
	GPT-2	24.01	5.78	21.15
	Biased TextRank	30.90	10.39	26.22
Vajdecka 2022 (VSE)	T5 Baseline	38.12	18.90	35.71
	SBERT+ LOF+T5 (13 % of sentences removed)	38.35	18.88	35.88
	Claim + T5 Baseline	39.19	20.56	36.92
	CLAIM + SBERT+LOF+T5 (13 % of sentences removed)	39.45	21.08	37.27
	CLAIM + SBERT+LOF+T5 (11 % of sentences removed)	39.76	21.37	37.54
	CLAIM + SBERT fine-tuned +LOF+T5 (13 % of sentences removed)	**40.76**	**22.00**	**38.36**
	CLAIM + SBERT fine-tuned +LOF+T5 (11 % of sentences removed)	39.55	20.69	37.11
	CLAIM + MORPHODITA + TF-IDF+LOF+T5 (13 % of sentences removed)	39.91	20.62	37.40
	CLAIM +MORPHODITA + TF-IDF+LOF+T5 (11 % of sentences removed)	39.86	20.59	37.30
	CLAIM + MORPHODITA + DOC2VEC+LOF+T5 (13 % of sentences removed)	38.58	19.62	36.20
	CLAIM + MORPHODITA + DOC2VEC+LOF+T5 (11 % of sentences removed)	39.04	20.65	36.70

Table 8: Politifact results

certain claim and on the y-axis we see the frequency of these tokens. We see that the graphs are almost identical.



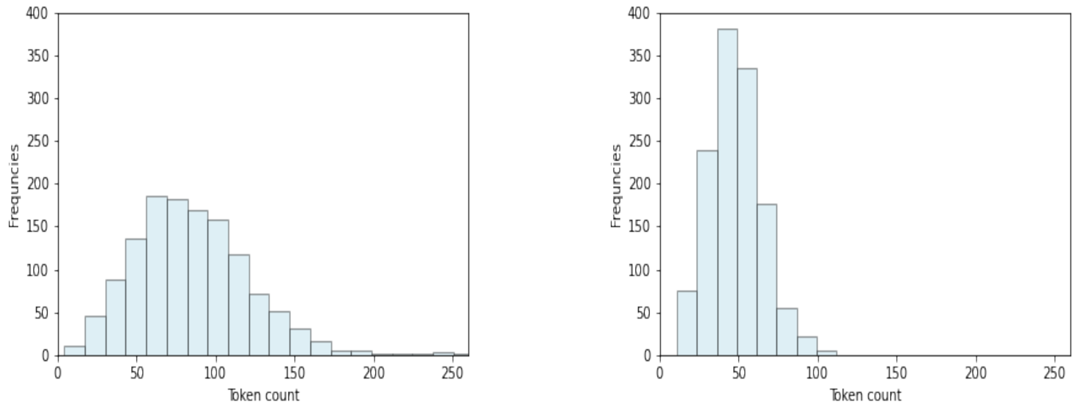
Ruling comments



Extractive summaries of ruling comments

Figure 22: Distribution of token counts (focusing on extractive summaries of the Politifact dataset)

Next, when we create abstractive summaries from the extractive summaries in the whole test data, we can visualize automatically generated summaries by their token count distribution (right graph in Figure 23). In Figure 23 we see the significant differences between justification (left graph in Figure 23) and the corresponding automatic summaries based on token count. In the left graph in Figure 23, we did not visualize three outliers as justification because they were more than 260 tokens long. However, even as we are outperforming the other procedures, summaries are still a bit shorter to adequate justifications in general.



Justifications Automatic summaries

Figure 23: Distribution of token counts (focusing on abstractive summaries of the Politifact dataset)

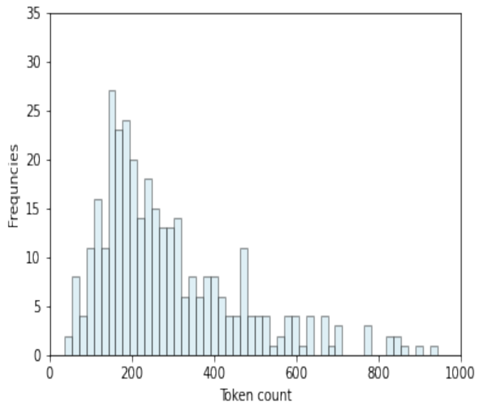
6.5.2 Demagog results

Similarly to Politifact, it has been shown that the concatenation of a claim together with an extractive summary promotes a higher-quality abstractive summary generated by T5 in comparison to fine-tuned *NLG* model without the claim included. As we can see in the results in Figure 9, the best model is the fine-tuned SENTENCE-BERT, but surprisingly very comparable results reached the usual TF-IDF. For this dataset, we were able to remove up to **24** percent of the sentences from the entire dataset.

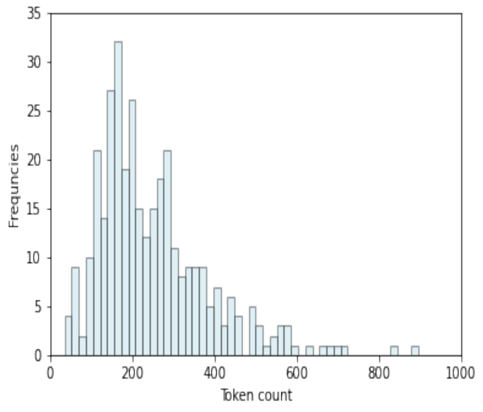
	Test set								
System	**ROUGE**$_{RAW-1}$			**ROUGE**$_{RAW-2}$			**ROUGE**$_{RAW-L}$		
	P	R	F	P	R	F	P	R	F
T5 Baseline	31.10	17.84	21.53	11.38	6.54	7.83	24.78	14.42	17.29
Claim + T5 Baseline	31.16	18.35	22.08	11.80	6.79	8.23	24.80	14.86	17.73
Claim + SBERT + LOF + T5 (24 % of sentences removed)	31.95	17.33	21.43	12.01	6.31	7.82	25.30	13.85	17.04
Claim + SBERT fine-tuned + LOF + T5 (24 % of sentences removed)	**32.73**	18.75	22.66	**12.97**	7.23	**8.82**	**26.29**	15.11	**18.25**
Claim + TF-IDF + LOF + T5 (24 % of sentences removed)	30.58	**19.92**	**23.08**	11.70	**7.51**	8.74	24.03	**15.82**	18.24
Claim + DOC2VEC + LOF + T5 (24 % of sentences removed)	31.41	16.89	20.82	11.50	6.06	7.49	25.29	13.78	16.89

Table 9: Demagog results

Now if we turn our attention to Figure 24, the removal of nearly one quarter (24 % of sentences) in the Demagog data will slightly distribute the frequency of token length in the test part. Before the extractive summary process, there was an outlier which was more than 1000 tokens long. This outlier as ruling comments is not include in the visualization. However, after the extractive summary had been processed, the length of this outlier was reduced to less than 1000 tokens (see right graph on Figure 24).



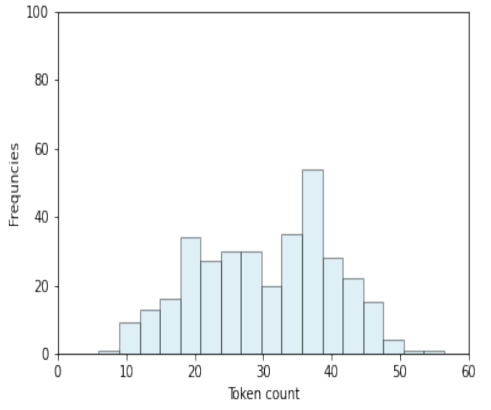
Ruling comments



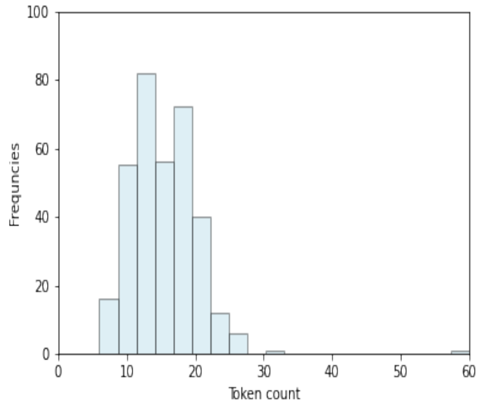
Extractive summaries of ruling comments

Figure 24: Distribution of token counts (focusing on extractive summaries of the Demagog dataset)

Similarly to Politifact, in Figure 25, we can see that if we compare the distribution of Justification frequency by its token lengths with distribution of generated summaries frequency by its token length, again the lengths of the generated summaries are shorter. In the abstractive summaries, we have the only one justification longer than 60 tokens in the test part, which is not included in the graph on the left side of Figure 25. The relevant generated summary was less than 60 tokens long, i.e. the right graph of the Figure 25 visualize all the summaries from testing set. We can see that the token distribution in the case of automatic summary length is again shorter.



Justifications



Automatic summaries

Figure 25: Distribution of token counts (focusing on abstractive summaries of the Demagog dataset)

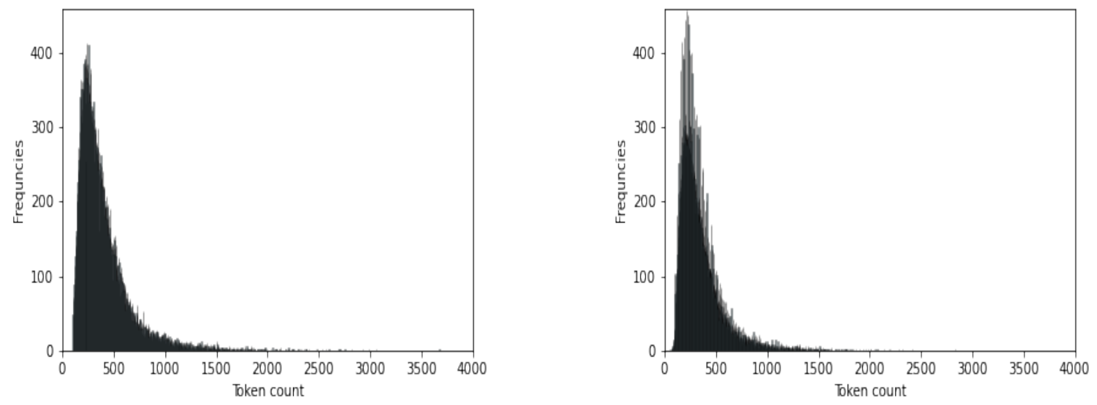
6.5.3 SumeCzech results

On the testing set of SumeCzech data set, we were able to improve Baseline T5 by removing 16 percent of the sentences from the whole data (see Table 10). Furthermore, we achieved the best results in **10** metrics out of 18 metrics, which no other method could reach.

Text → Headline																			
		Test set									Out-of-domain test set								
Source	System	**ROUGE**$_{RAW-1}$			**ROUGE**$_{RAW-2}$			**ROUGE**$_{RAW-L}$			**ROUGE**$_{RAW-1}$			**ROUGE**$_{RAW-2}$			**ROUGE**$_{RAW-L}$		
		P	R	F	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
SumeCzech (Straka et al., 2018)	first	7.4	**13.5**	8.9	1.1	2.2	1.3	6.5	11.7	7.7	6.7	13.6	8.3	1.3	2.8	1.6	5.9	**12.0**	7.4
	random	5.9	10.3	6.9	0.5	1.0	0.6	5.2	8.9	6.0	5.2	10.0	6.3	0.6	1.4	0.8	4.6	8.9	5.6
	textrank	6.0	16.5	8.3	0.8	2.3	1.1	5.0	13.8	6.9	5.8	**16.9**	8.1	1.1	3.4	1.5	5.0	14.5	6.9
	tensor2tensor	8.8	7.0	7.5	0.8	0.6	0.7	8.1	6.5	7.0	6.3	5.1	5.5	0.5	0.4	0.4	5.9	4.8	5.1
Named entities (Marek et al., 2021)	Seq2Seq	16.1	14.1	**14.6**	2.5	2.1	2.2	14.6	**12.8**	13.2	13.1	11.8	12	2	1.7	1.8	12.1	11	11.2
	Seq2Seq–NER	**16.2**	14.1	14.7	2.5	2.1	2.2	**14.7**	**12.8**	**13.3**	13.7	11.9	12.4	2	1.7	1.8	12.6	11.1	11.4
Peter Vajdecka (only 10 % of training data)	T5	15.4	11.0	12.5	3.2	2.3	2.6	14.2	10.1	11.5	15.9	11.9	13.2	4.4	3.2	3.6	14.9	11.2	12.4
	T5-SBERT-LOF	15.8	11.4	12.9	**3.5**	**2.5**	**2.8**	14.6	10.6	11.9	**16.5**	12.4	**13.7**	**4.8**	**3.5**	**3.9**	**15.4**	11.6	**12.9**

Table 10: SumeCzech results

The figure 26 again shows the distribution of tokens before (graph to the left) and after (graph to the right) the extractive summary is processed. we are focusing on out the best model (see Figure 10) where 16 percent of sentences are deleted from the entire dataset. From the Figure 26, comparing the most frequent vertices of the graphs, we can see that many sentences have been removed (after the extractive summary is created) from reports with the most frequent token lengths. In both graphs we have outliers as text or an extractive text summary with a length of more than 4000 tokens. From Figure 26, the left graph is missing total of 12 outliers and the graph on the right (extractive summaries) only 3 outliers are not visualized.

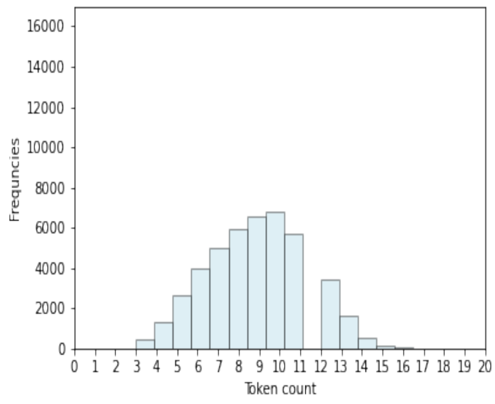


Texts

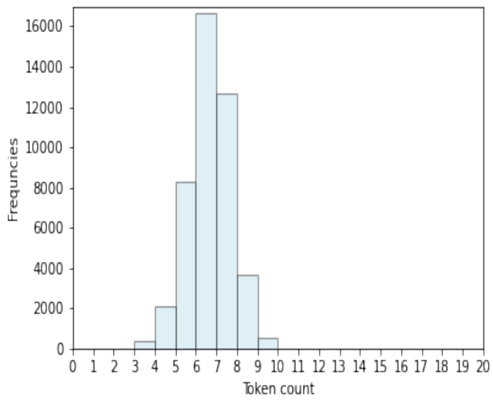
Extractive summaries of text

Figure 26: Distribution of token counts (focusing on extractive summaries of the SumeCzech dataset)

The abstractive summarization, even after extractive summaries are created, seems to achieve much shorter results. In Figure 27 we see that the most frequent number of „gold“ headlines is approximately 10 tokens. In the automatic summary we only got the most frequent number of tokens for the generated count 6. Thus, even with short texts, generated summaries are shorter in comparison to correct Headlines. In the case of generating headlines, we consider as outliers those summaries that have a length of more than 20 tokens. They were only found in headlines three times and we do not object them in Figure 27.



Headlines



Automatic short summaries

Figure 27: Distribution of token counts (focusing on abstractive summaries of the SumeCzech dataset)

6.6 Summaries of the best method

In this section we will show a couple of concrete text examples, where the best and the worst generated summaries are sorted according to the **ROUGE-2 F1 score** for the English dataset (analogous to the **ROUGERAW-2 F1 score** for the Czech dataset). The reason why we choose *ROUGE-2 F1 score* and not other kind of *ROUGE-N F1 score* or *ROUGE-L F1 score*, is actually paper (Atanasova et al., 2020), where the authors found *ROUGE-2 F1 score* the most valuable when comparing similar texts.

6.6.1 Politifact examples

For fine-tuning of the best model, the value of the **max words summary** hyperparameter is **150 tokens** and the value for the **max words text** hyperparameter is **1200** tokens. We use batch size of **9** training pairs. We fine-tuned T5 transformer on 3 epochs and SENTENCE-BERT was fine-tuned on 5 epochs.

So that the overall results are not just about numbers, we have decided to display a few examples of the best summaries and a few examples of the worst summaries. Three of the best examples are shown in Table 11. As we see in the first row of this table, the model is so intelligent that if the claim is true it will tell us the final result immediately. That is, in this case of first automatic summary, what „Warren“ actually said. In the case of the second summary, we can see perfectly well that the idea of justification but also of automatic summary is exactly the same and we can say that automatic summary is a paraphrase of justification. And in the case of the last claim in the Table 11 we see a beautiful paraphrase between justification and the automatic summary, so that "here was no massacre." and "it wasn't a massacre" indicating that the statement is false.

Claim	**Justification**	**Automatic summary**
Look at the men on this stage. Collectively, they have lost 10 elections. The only people on this stage who have won every single election that they've been in are the women, Amy and me.	Warren said, Look at the men on this stage. Collectively, they have lost 10 elections. The only people on this stage who have won every single election that they've been in are the women, Amy and me. We checked the electoral histories of the candidates on stage and found that Warren spoke accurately.	Warren said, Collectively, they have lost 10 elections. The only people on this stage who have won every single election that they've been in are the women, Amy and me.
Says there have been some job gains in the McMansion State since Mr. Christie took office, but they have lagged gains both in the nation as a whole and in New York and Connecticut, the obvious points of comparison.	In his May 28 column, Krugman claimed "there have been some job gains in the McMansion State since Mr. Christie took office, but they have lagged gains both in the nation as a whole and in New York and Connecticut, the obvious points of comparison." Krugman's analysis is correct. In terms of the percentage increases in total jobs and private-sector jobs as of April 2012, New Jersey's growth since the beginning of Christie's tenure has been less than the increases in New York, Connecticut and the nation as a whole. To comment on this ruling, go to NJ.com.	Krugman said, "There have been some job gains in the McMansion State since Mr. Christie took office, but they have lagged gains both in the nation as a whole and in New York and Connecticut, the obvious points of comparison." In terms of total jobs and private-sector job growth since January 2010, New Jersey's overall job growth has been less than New York, Connecticut and the nation as a whole during the same period. To comment on this ruling, go to NJ.com.
Two Iraqis came here to this country, were radicalized and they were the masterminds behind the Bowling Green massacre. … It didn't get covered	Conway said that "two Iraqis came here to this country, were radicalized and they were the masterminds behind the Bowling Green massacre. … It didn't get covered." Put simply, there was no massacre.	Conway said that "two Iraqis came here to this country, were radicalized and they were the masterminds behind the Bowling Green massacre.... It didn't get covered." There was indeed a terrorism-related situation in Bowling Green, Ky., but it wasn't a massacre.

Table 11: Examples of the best summaries on the Politifact dataset

On the other hand, if we look at the worst examples of the summaries shown in Table 12, we see that in fact the automatic summaries are of course not identical to the Justification, but it is actually a paraphrase of the Justification, which is actually correct. Of course there are sometimes mistakes, but finally the results of the automatic summary are surprisingly inspiring.

Claim	Justification	Automatic summary
In South Dakota, I'm putting prayer back in our schools.	No specific policy exists, nor does any specific proposal or a timeline for producing one. That could change if Noem produces a proposal, the Legislature approves it, and it passes muster in court.	Noem told the crowd, In South Dakota, I'm putting prayer back in our schools. The Supreme Court ruled that schools and their representatives could not lead organized prayers in public schools. However, it does not prohibit silent, private prayer in schools.
Between 2000 and 2010, the illegal immigrant population of Texas increased 60 percent, while California and Florida had no increase.	Romney is essentially correct on the numbers, but his strong implication that Perry is responsible is a significant stretch.	Romney said, "California and Florida have had 60 percent increase in illegal immigration." The numbers for Texas and California are broadly supported by the best estimates available. However, they are not as reliable as Romney's statement suggests.
Says under Mitt Romney, Massachusetts was 47th in job creation.	The job-creation ranking for Massachusetts is right; the implication that Romney is solely to blame for the state's low standing does not hold up.	Castro said that under Romney, Massachusetts was 47th in job creation. The state ranked 47th in the nation in job creation.

Table 12: Examples of the worst summaries on the Politifact dataset

6.6.2 Demagog examples

For fine-tuning of the best model, the value of the **max words summary** hyperparameter is **50 tokens** and the value for the **max words text** hyperparameter is **800** tokens. We use batch size of **12** training pairs. We fine-tuned T5 transformer on 3 epochs and SENTENCE-BERT was fine-tuned on 5 epochs.

Table 13 shows that the automatic summaries for the Czech Demagog are no worse in comparison to Politifact. The first automatic summary is indeed with identical sense as justification, the second automatic summary is not complete (missing „Ministerstvo pro místní rozvoj“). In the third one the entity of the date is different but intention of reference and generated summary is same. The fourth generated sub-file is complete, but the time of the end of the year is missing.

Claim	Justification	Automatic summary
Prosadíme novelu zákona o důchodovém pojištění, která zvýší základní výměru důchodu na 10 % průměrné mzdy.	Novela zákona o důchodovém pojištění, která zvýšila výměru důchodu na 10 % průměrné mzdy, již prošla legislativním procesem a příslušná ustanovení jsou od 1. ledna 2019 účinná.	Novela zákona o důchodovém pojištění, která zvýší základní výměru důchodu na 10 % průměrné mzdy, je účinná od 1. ledna 2019.
Vybírá to prostě nezávislá komise (města pro zavedení 5G, pozn. Demagog.cz).	Soutěž 5G pro 5 měst vyhlásilo společně Ministerstvo průmyslu a obchodu a Ministerstvo pro místní rozvoj v druhé polovině října. Složení, a tedy míra nezávislosti komise, nebyla v době analýzy výroku dohledatelná.	Soutěž 5G pro 5 měst vyhlásilo Ministerstvo průmyslu a obchodu, Ministerstvo pro místní rozvoj, Ministerstvo průmyslu a obchod
Začali jsme testovat od 23. ledna, tehdy tady ještě nebyl žádný případ.	K prvnímu dohledatelnému testování na onemocnění COVID-19 v České republice došlo 24. ledna 2020. V té době v ČR skutečně nebyl žádný případ infikování koronavirem.	V České republice 23. ledna 2020 nebyl žádný případ onemocnění COVID-19.
(...) tady vzniká společná iniciativa SPD a Pirátů na to, abychom se těmi platy zabývali.	Piráti a SPD požádali, aby mimořádná schůze kvůli projednání novely zákona o platech politiků proběhla ještě do konce roku 2018.	Piráti a SPD požádali o mimořádnou schůzi kvůli dalšímu projednání novely zákona o platech ústavních činitelů.

Table 13: Examples of the best summaries on the Demagog dataset

If we focus on the the worst automatic summaries in Demagog data set in Figure 14, the first automatic summary is actually a simpler paraphrase of justification. The second summary is exactly the same as the first. The other two seem to have some errors. In the first case it is obviously an entity, in the other case it is a very general paraphrase.

Claim	Justification	Automatic summary
TAKÁČ, moderátor: A Evropská unie, Evropská komise to ocenila? Jste říkal. To znamená nějaká filozofická výhrada typu: jádro ne, to nepřipadá v úvahu, ta nezazněla? HAVLÍČEK: V žádném případě vůbec nezazněla. V tomto naopak zaznělo to, že je správně, že míříme k bezemisní ekonomice, respektive energetice ad 1 a ad 2 jasně zaznělo to, že jádro je bezemisním zdrojem. To, že někdo má tendenci čas od času, zejména z řad těch kritiků, to zpochybňovat, tak toto zaznělo doslova a do písmene od generální ředitelky, (...) tak to řekla přímo generální ředitelka generálního ředitelství Energo, Evropské komise.	Dne 20. listopadu 2019 jednala česká delegace MPO v Bruselu o budoucí podobě energetiky ČR a roli jádra v ní. Náměstek pro energetiku na MPO slova ministra Havlíčka potvrdil.	Česká republika mluví o výstavbě nového jaderného bloku v Dukovany.
Pak se stalo něco, čemu nerozumím, že paní ministryně (Maláčová, pozn. Demagog.cz) přestala chodit na výbory, začala posílat ředitele odborů, což je proti jednacímu řádu, takže jsme přerušili jednání. Nakonec tam tedy začali chodit kompetentní náměstci.	Zmíněné chování ministryně Maláčové je skutečně proti jednacímu řádu Poslanecké sněmovny. Ovšem popsaným způsobem se zachovala pouze jednou.	Ministryně Maláčová byla předsedkyní výboru pro sociální politiku. Na jednání však dorazila pouze jednou, na kterých nebyla přítomena náměstka.
Dobrá zpráva – Evropská rada se konečně na něčem dokázala shodnout. Špatná zpráva – v porovnání s návrhem Komise z května je rozpočet na výzkum ponížený o 14 miliard €. Pěkně prosím, škrtat na výzkumu není šetřivé, ale prostě hloupé.	Evropská rada se 21. července dohodla na snížení rozpočtu na vědu a výzkum proti původnímu plánu o 25,17 mld. eur v kapitole Výzkum. K dalším škrtům pak došlo v kapitolách Evropského obranného fondu či Evropského kosmického programu.	V květnu došlo k přijetí dohody na víceletém finančnímu rámci. Rozpočet na výzkum byl zvýšen o 13,7 mld. eur.
V mediálním prostoru se opakovaně řeší otázka vyhošťování diplomatů, jaké jsou důvody podle Vídeňské úmluvy o diplomatických vztazích. Je to skutečně pouze řada konkrétních důvodů, které můžou hostující stranu vést k vyhoštění diplomatů.	Vídeňská úmluva o diplomatických stycích umožňuje označit jakéhokoli diplomata jiného státu za personu non grata a tím fakticky ukončit jeho diplomatickou misi. Může se tak ovšem stát i bez udání důvodu.	V médiálním prostoru se opakovaně řeší otázka vyhošťovování diplomatů.

Table 14: Examples of the worst summaries on the Demagog dataset

6.6.3 SumeCzech examples

For fine-tuning of the best model the value of the **max words summary** hyperparameter is **15 tokens** and the value for the **max words text** hyperparameter is **500** tokens. We use batch size of **17** training pairs. We fine-tuned T5 transformer on 3 epochs.

The results of the best summaries are shown in Table 15. So the first three generated headlines are identical to the reference headlines (pretty fascinating). In the fourth generated headline there is only an extending word „jsou“ compared to reference summary. In the fifth automatic headline there is missing of subject „vysnené bydleni“ and a similar problem in the last summary (see Table 15).

Headline	**Automatic headline**
Metal Gear Solid: Ground Zeroes	Metal Gear Solid: Ground Zeroes
Lázně Poděbrady jsou nejen na srdce	Lázně Poděbrady jsou nejen na srdce
Trains Trucks Tycoon - demo	Trains Trucks Tycoon - demo
Nejčastější zhoubné nádory u mužů	Nejčastější zhoubné nádory u mužů jsou
Rodinné domy v Unhošti poskytly sedmi rodinám vysněné bydlení	Rodinné domy v Unhošti poskytly sedmi rodinám
Neocron Arcade: The N.M.E. Project - kvalitní akční řežba	Neocron Arcade: The N.M.E. Project

Table 15: Examples of the best summaries on the SumeCzech dataset

In Table 16, we picture the worst automatic summaries using testing set of SumeCzech dataset. Since they are actually shorter summaries on average compared to the previous two datasets, it can be seen that these summaries contain real fabrications. We can see that there is not even a paraphrase. Since the headlines have specific word formation, the fine-tuning is insufficient at the ends of the generated summaries. And headline punching is an overall problem, even though the model achieves the excellent results.

Headline	**Automatic headline**
Makro jde blíž podnikatelům. Přejmenovává se na Vy Makro	Obchody se zajímají o malé a střed
Ojedinělá kniha o antické technologii Z východního Halabu od soboty uteklo 10 tisíc lidí	Antická trilogie Jiřího Rakušana V Sýrii se chystá poslední exodus, tvrd
Kupte hrachovku, zvolte Jurečku! Kandidát se nabízí v letácích obchodu	Poslanci chtějí ušetřit statisíce za hlas
Politici perlili: Dubové, prempos a odkloňování	Poslanci zvolili vtipného europoslance
Spolujezdec mrtvého pilota je mimo ohrožení života	Při nehodě u Korába zemřel spolujezdec

Table 16: Examples of the worst summaries on the SumeCzech dataset

Conclusion

The goal of this work was to combine natural language processing and natural language generation methods to create a hybrid procedure for generating abstract summaries from extraction summaries of fact-checking reports. For the abstract summarization we applied the T5 transformer, which we considered as a "black box", whose summarization we tried to improve through the methods we proposed in this work.

As the first method we proposed was an *extractive summarization* by means of *Local Outlier Factor*, it turned out that by properly determining the number of sentences to remove from the initial form of the report, we can improve the quality of the summary of T5 transformer. In the work we could have two following possibilities of focusing:

1. trying to find the ideal maximum number of sentences to remove, so that the performance of the T5 transformer is neither improved nor degraded;
2. to search for the optimal number of sentences to remove, so as to achieve the best possible transformer summary results.

We decided to focus on the second kind of alternative because we wanted to compare our work with the results of other experts on the data used.

The *proposed extractive summarization* algorithm was applied to three different datasets. It turned out that in each dataset we can use this procedure to improve abstractive T5 summaries with reduced report length by its sentences, both in English and in Czech.

As soon as we confirmed that the proposed extractive summarization definitely improves the results of the T5 transformer, our next goal was to improve the vector forms of the sentences in the reports so that the proposed extractive method achieves even better quality. Our initial vectorization method was SENTENCE-BERT, which was further improved by another proposed algorithm to generate special data in the form of triplets. These triplets were used to fine-tune the new Sentence-BERT model for generating extractive fact-checking summaries in English and Czech. These extractive summaries were reapplied to the T5 Transformer. Thus, we created two fine-tuned SENTENCE-BERT models, one for English language and another for Czech language.

The proposed triplet extraction method using the fine-tuned SENTENCE-BERT achieved the best results compared to other embeddings such as different variants of SENTENCE-BERT, DOC2VEC or TF-IDF. The presence and testing of different types of embeddings did not only confirm that the specially fine-tuned SENTENCE-BERT achieves the best results out of them, but that the extraction method itself is independent of vectorization type.

Surprisingly, especially in Czech, the vectorization with the primitive TF-IDF, where the weights were based on the lemmatizer MORPHODITA, achieved very similar results to the SENTENCE-BERT. In

Czech, five test metrics were in favor of the fine-tuned SENTENCE-BERT and 4 in favor of TF-IDF. In English, the fine-tuned SENTENCE-BERT was clearly the best performer across all metrics examined, but TF-IDF were not far from fine-tuned SENTENCE-BERT.

We also showed that extractive summarization improves the summary results when removing **13** % of the sentences of the English dataset, regardless of the type of embeddings. In the Czech dataset, we showed that when removing up to **24** percent of the sentences, only our specially trained SENTENCE-BERT can improve the summary results, and in the case of using the general not fine-tuned SENTENCE-BERT, it even degrades the summary results.

The paper showed that T5 has yet another shortcoming, like other neural network models for text generation, namely the shorter length of the generated text. Although we have overcome many of the state-of-the-art results, the abstract model tends to generate shorter summaries in general. And this is definitely something we should focus on in the future. On the other hand, we have shown that some of the worse results according to the *ROUGE 2 F score* are not that bad, since T5 paraphrases and mostly preserves the intent of the generated summary.

Bibliography

ALHINDI, Tariq; PETRIDIS, Savvas; MURESAN, Smaranda, 2018. Where is your evidence: improving fact-checking by justification modeling. In: *Proceedings of the first workshop on fact extraction and verification (FEVER)*, pp. 85–90.

ALOMARI, Ayham; IDRIS, Norisma; SABRI, Aznul Qalid Md; ALSMADI, Izzat, 2022. Deep reinforcement and transfer learning for abstractive text summarization: A review. *Computer Speech & Language*. Vol. 71, p. 101276.

ATANASOVA, Pepa; SIMONSEN, Jakob Grue; LIOMA, Christina; AUGENSTEIN, Isabelle, 2020. Generating fact checking explanations. *arXiv preprint arXiv:2004.05773*.

BA, Jimmy Lei; KIROS, Jamie Ryan; HINTON, Geoffrey E, 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

BAPNA, Ankur; ARIVAZHAGAN, Naveen; FIRAT, Orhan, 2019. Simple, scalable adaptation for neural machine translation. *arXiv preprint arXiv:1909.08478*.

BEHRENDT, Maike; HARMELING, Stefan, 2021. ArgueBERT: How To Improve BERT Embeddings for Measuring the Similarity of Arguments. In: *Proceedings of the 17th Conference on Natural Language Processing (KONVENS 2021)*, pp. 28–36.

BHAT, Iram Khurshid; MOHD, Mudasir; HASHMY, Rana, 2018. Sumitup: A hybrid single-document text summarizer. In: *Soft computing: Theories and applications*. Springer, pp. 619–634.

BREUNIG, Markus M; KRIEGEL, Hans-Peter; NG, Raymond T; SANDER, Jörg, 2000. LOF: identifying density-based local outliers. In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pp. 93–104.

CARVALLO, Andres; PARRA, Denis; LOBEL, Hans; SOTO, Alvaro, 2020. Automatic document screening of medical literature using word and text embeddings in an active learning setting. *Scientometrics*. Vol. 125, no. 3, pp. 3047–3084.

DEVLIN, Jacob; CHANG, Ming-Wei; LEE, Kenton; TOUTANOVA, Kristina, 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

DING, Hanxiang; DING, Kun; ZHANG, Jingwei; WANG, Yue; GAO, Lie; LI, Yuanliang; CHEN, Fudong; SHAO, Zhixiong; LAI, Wanbin, 2018. Local outlier factor-based fault detection and evaluation of photovoltaic system. *Solar Energy*. Vol. 164, pp. 139–148.

FLORIDI, Luciano; CHIRIATTI, Massimo, 2020. GPT-3: Its nature, scope, limits, and consequences. *Minds and Machines*. Vol. 30, no. 4, pp. 681–694.

GARDNER, William A, 1984. Learning characteristics of stochastic-gradient-descent algorithms: A general study, analysis, and critique. *Signal processing*. Vol. 6, no. 2, pp. 113–133.

GARG, Apar; ADUSUMILLI, Saiteja; YENNETI, Shanmukha; BADAL, Tapas; GARG, Deepak; PANDEY, Vivek; NIGAM, Abhishek; GUPTA, Yashu Kant; MITTAL, Gyan; AGARWAL, Rahul, 2020. NEWS Article Summarization with Pretrained Transformer. In: *International Advanced Computing Conference*, pp. 203–211.

GEHRMANN, Sebastian; ADEWUMI, Tosin; AGGARWAL, Karmanya; AMMANAMANCHI, Pawan Sasanka; ANUOLUWAPO, Aremu; BOSSELUT, Antoine; CHANDU, Khyathi Raghavi; CLINCIU, Miruna; DAS, Dipanjan; DHOLE, Kaustubh D, et al., 2021. The gem benchmark: Natural language generation, its evaluation and metrics. *arXiv preprint arXiv:2102.01672*.

GUPTA, Anushka; CHUGH, Diksha; KATARYA, Rahul, et al., 2021. Automated News Summarization Using Transformers. *arXiv preprint arXiv:2108.01064*.

HANSELOWSKI, Andreas; STAB, Christian; SCHULZ, Claudia; LI, Zile; GUREVYCH, Iryna, 2019. A richly annotated corpus for different tasks in automated fact-checking. *arXiv preprint arXiv:1911.01214*.

HASSAN, Naeemul; ADAIR, Bill; HAMILTON, James T; LI, Chengkai; TREMAYNE, Mark; YANG, Jun; YU, Cong, 2015. The quest to automate fact-checking. In: *Proceedings of the 2015 computation+ journalism symposium*.

HASSAN, Naeemul; ARSLAN, Fatma; LI, Chengkai; TREMAYNE, Mark, 2017. Toward automated fact-checking: Detecting check-worthy factual claims by claimbuster. In: *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1803–1812.

HASSANI, Ali; WALTON, Steven; SHAH, Nikhil; ABUDUWEILI, Abulikemu; LI, Jiachen; SHI, Humphrey, 2021. Escaping the big data paradigm with compact transformers. *arXiv preprint arXiv:2104.05704*.

HOCHREITER, Sepp, 1998. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*. Vol. 6, no. 02, pp. 107–116.

HOULSBY, Neil; GIURGIU, Andrei; JASTRZEBSKI, Stanislaw; MORRONE, Bruna; DE LAROUSSILHE, Quentin; GESMUNDO, Andrea; ATTARIYAN, Mona; GELLY, Sylvain, 2019. Parameter-efficient transfer learning for NLP. In: *International Conference on Machine Learning*, pp. 2790–2799.

HOWARD, Jeremy; RUDER, Sebastian, 2018. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*.

JEON, Daeseong; AHN, Joon Mo; KIM, Juram; LEE, Changyong, 2022. A doc2vec and local outlier factor approach to measuring the novelty of patents. *Technological Forecasting and Social Change*. Vol. 174, p. 121294.

JOWETT, Gart S; O'DONNELL, Victoria, 2006. *What is Propaganda and How does It Differ From Propaganda*. London: Sage Publication.

EL-KASSAS, Wafaa S; SALAMA, Cherif R; RAFEA, Ahmed A; MOHAMED, Hoda K, 2021. Automatic text summarization: A comprehensive survey. *Expert Systems with Applications*. Vol. 165, p. 113679.

KAZEMI, Ashkan; LI, Zehua; PÉREZ-ROSAS, Verónica; MIHALCEA, Rada, 2021. Extractive and Abstractive Explanations for Fact-Checking and Evaluation of News. *arXiv preprint arXiv:2104.12918*.

KIM, Donghwa; SEO, Deokseong; CHO, Suhyoun; KANG, Pilsung, 2019. Multi-co-training for document classification using various document representations: TF–IDF, LDA, and Doc2Vec. *Information Sciences*. Vol. 477, pp. 15–29.

LAU, Jey Han; BALDWIN, Timothy, 2016. An empirical evaluation of doc2vec with practical insights into document embedding generation. *arXiv preprint arXiv:1607.05368*.

LE, Quoc; MIKOLOV, Tomas, 2014. Distributed representations of sentences and documents. In: *International conference on machine learning*, pp. 1188–1196.

LIN, Chin-Yew, 2004. Rouge: A package for automatic evaluation of summaries. In: *Text summarization branches out*, pp. 74–81.

LIU, Pengfei; QIU, Xipeng; HUANG, Xuanjing, 2016. Recurrent neural network for text classification with multi-task learning. *arXiv preprint arXiv:1605.05101*.

MAREK, Petr; MÜLLER, Štěpán; KONRÁD, Jakub; LORENC, Petr; PICHL, Jan; ŠEDIV, Jan, 2021. Text Summarization of Czech News Articles Using Named Entities. *arXiv preprint arXiv:2104.10454*.

MIKOLOV, Tomas; CHEN, Kai; CORRADO, Greg; DEAN, Jeffrey, 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

MIKOLOV, Tomas; JOULIN, Armand; CHOPRA, Sumit; MATHIEU, Michael; RANZATO, Marc'Aurelio, 2014. Learning longer memory in recurrent neural networks. *arXiv preprint arXiv:1412.7753*.

MIKOLOV, Tomas; SUTSKEVER, Ilya; CHEN, Kai; CORRADO, Greg S; DEAN, Jeff, 2013. Distributed representations of words and phrases and their compositionality. In: *Advances in neural information processing systems*, pp. 3111–3119.

NAKOV, Preslav; DA SAN MARTINO, Giovanni; ELSAYED, Tamer; BARRÓN-CEDENO, Alberto; MIGUEZ, Rubén; SHAAR, Shaden; ALAM, Firoj; HAOUARI, Fatima; HASANAIN, Maram; BABULKOV, Nikolay, et al., 2021. The CLEF-2021 CheckThat! lab on detecting check-worthy claims, previously fact-checked claims, and fake news. In: *European Conference on Information Retrieval*, pp. 639–649.

PHILIPP, George; SONG, Dawn; CARBONELL, Jaime G, 2017. The exploding gradient problem demystified-definition, prevalence, impact, origin, tradeoffs, and solutions. *arXiv preprint arXiv:1712.05577*.

PŘIBÁŇ, Pavel; HERCIG, Tomáš; STEINBERGER, Josef, 2019. Machine Learning Approach to Fact-Checking in West Slavic Languages. In: *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2019)*, pp. 973–979.

RAFFEL, Colin; SHAZEER, Noam; ROBERTS, Adam; LEE, Katherine; NARANG, Sharan; MATENA, Michael; ZHOU, Yanqi; LI, Wei; LIU, Peter J, 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.

REIMERS, Nils; GUREVYCH, Iryna, 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

REN, Fuji; XUE, Siyuan, 2020. Intention detection based on siamese neural network with triplet loss. *IEEE Access*. Vol. 8, pp. 82242–82254.

RONG, Xin, 2014. word2vec parameter learning explained. *arXiv preprint arXiv:1411.2738*.

SALMINEN, Joni; HOPF, Maximilian; CHOWDHURY, Shammur A; JUNG, Soon-gyo; ALMEREKHI, Hind; JANSEN, Bernard J, 2020. Developing an online hate classifier for multiple social media platforms. *Human-centric Computing and Information Sciences*. Vol. 10, no. 1, pp. 1–34.

SALTON, Gerard; BUCKLEY, Christopher, 1988. Term-weighting approaches in automatic text retrieval. *Information processing & management*. Vol. 24, no. 5, pp. 513–523.

SCHROFF, Florian; KALENICHENKO, Dmitry; PHILBIN, James, 2015. Facenet: A unified embedding for face recognition and clustering. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 815–823.

SEO, Seungwan; SEO, Deokseong; JANG, Myeongjun; JEONG, Jaeyun; KANG, Pilsung, 2020. Unusual customer response identification and visualization based on text mining and anomaly detection. *Expert Systems with Applications*. Vol. 144, p. 113111.

SHAAR, Shaden; MARTINO, Giovanni Da San; BABULKOV, Nikolay; NAKOV, Preslav, 2020. That is a known lie: Detecting previously fact-checked claims. *arXiv preprint arXiv:2005.06058*.

SHI, Baoxu; WENINGER, Tim, 2016. Discriminative predicate path mining for fact checking in knowledge graphs. *Knowledge-based systems*. Vol. 104, pp. 123–133.

SINGH, Anita Kumari; SHASHI, Mogalla, 2019. Vectorization of text documents for identifying unifiable news articles. *Int. J. Adv. Comput. Sci. Appl*. Vol. 10, no. 7.

SRIVASTAVA, Nitish; HINTON, Geoffrey; KRIZHEVSKY, Alex; SUTSKEVER, Ilya; SALAKHUTDINOV, Ruslan, 2014. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*. Vol. 15, no. 1, pp. 1929–1958.

STRAKA, Milan; MEDIANKIN, Nikita; KOCMI, Tom; ŽABOKRTSK, Zdeněk; HUDEČEK, Vojtěch; HAJIC, Jan, 2018. Sumeczech: Large czech news-based summarization dataset. In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*.

THORNE, James; VLACHOS, Andreas; COCARASCU, Oana; CHRISTODOULOPOULOS, Christos; MITTAL, Arpit, 2019. Proceedings of the Second Workshop on Fact Extraction and VERification (FEVER). In: *Proceedings of the Second Workshop on Fact Extraction and VERification (FEVER)*.

VARGAS-CALDERÓN, Vladimir; CAMARGO, Jorge E, 2019. Characterization of citizens using word2vec and latent topic analysis in a large set of tweets. *cities*. Vol. 92, pp. 187–196.

VASWANI, Ashish; SHAZEER, Noam; PARMAR, Niki; USZKOREIT, Jakob; JONES, Llion; GOMEZ, Aidan N; KAISER, Łukasz; POLOSUKHIN, Illia, 2017. Attention is all you need. *Advances in neural information processing systems*. Vol. 30.

VLACHOS, Andreas; RIEDEL, Sebastian, 2014. Fact checking: Task definition and dataset construction. In: *Proceedings of the ACL 2014 workshop on language technologies and computational social science*, pp. 18–22.

VOSOUGHI, Soroush; ROY, Deb; ARAL, Sinan, 2018. The spread of true and false news online. *Science*. Vol. 359, no. 6380, pp. 1146–1151.

WALKOWIAK, Tomasz, 2021. Subject Classification of Texts in Polish-from TF-IDF to Transformers. In: *International Conference on Dependability and Complex Systems*, pp. 457–465.

WANG, William Yang, 2017. " liar, liar pants on fire": A new benchmark dataset for fake news detection. *arXiv preprint arXiv:1705.00648*.

WU, Yonghui; SCHUSTER, Mike; CHEN, Zhifeng; LE, Quoc V; NOROUZI, Mohammad; MACHEREY, Wolfgang; KRIKUN, Maxim; CAO, Yuan; GAO, Qin; MACHEREY, Klaus, et al., 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

XUE, Linting; CONSTANT, Noah; ROBERTS, Adam; KALE, Mihir; AL-RFOU, Rami; SIDDHANT, Aditya; BARUA, Aditya; RAFFEL, Colin, 2020. mT5: A massively multilingual pre-trained text-to-text transformer. *arXiv preprint arXiv:2010.11934*.

YOU, Lan; PENG, Qingxi; XIONG, Zenggang; HE, Du; QIU, Meikang; ZHANG, Xuemin, 2020. Integrating aspect analysis and local outlier factor for intelligent review spam detection. *Future Generation Computer Systems*. Vol. 102, pp. 163–172.

ZAHEER, Manzil; GURUGANESH, Guru; DUBEY, Kumar Avinava; AINSLIE, Joshua; ALBERTI, Chris; ONTANON, Santiago; PHAM, Philip; RAVULA, Anirudh; WANG, Qifan; YANG, Li, et al., 2020. Big bird: Transformers for longer sequences. *Advances in Neural Information Processing Systems*. Vol. 33, pp. 17283–17297.

ZHANG, Wei; WEI, Wei; WANG, Wen; JIN, Lingling; CAO, Zheng, 2021. Reducing bert computation by padding removal and curriculum learning. In: *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 90–92.