Jagiellonian University in Cracow
Faculty of Physics, Astronomy and Applied Computer Science

Uniwersytet Jagielloński w Krakowie
Wydział Fizyki, Astronomii i Informatyki Stosowanej

Wojciech Ozimek
Number of album: 1124802

Automatic summary of texts in Polish

Title in Polish:
Automatyczne podsumowywanie tekstów w języku polskim

Applied Computer Science
Specialization: General Computer Science

Master thesis written under scientific supervision of
Dr. Jeremi Ochab

Kraków, June 2020

Abstract

The automatic text summarizing task is one of the most complex problems in the field of natural language processing. In this dissertation, we present the abstraction-based summarization approach which allows to paraphrase the original text and generate new sentences. Creation of new formulations, completely different from the original text is similar to how humans summarize texts. To achieve this, we propose the deep learning method using *Sequence to Sequence* architecture with the attention mechanism. The goal is to create the model for Polish language, using dataset containing over 200,000 articles from Polish websites, split into text and summary parts. Presented outcomes look promising, obtaining decent results utilizing standard metrics for such type of task.

Based on review of prior research done during experiments, this is the very first attempt of applying abstractive text summarization techniques for Polish language.

Streszczenie

Automatyczne streszczanie tekstów jest jednym z najbardziej złożonych problemów w dziedzinie przetwarzania języka naturalnego. W niniejszej pracy przedstawię abstrakcyjne podejście do tworzenia streszczeń, które pozwala na parafrazowanie oryginalnego tekstu. Tworzenie zupełnie nowych sformułowań, niewystępujących w tekście wyjściowym, jest podobne do sposobu, w jaki ludzie tworzą streszczenia. By to osiągnąć, proponuję mechanizm oparty o metody głębokiego uczenia z wykorzystaniem architektury *Sequence to Sequence* wraz z mechanizmem uwagi. Celem jest stworzenie takiego mechanizmu dla języka polskiego, wykorzystując w tym celu zbiór danych składający się z ponad 200 000 artykułów z polskich stron internetowych, podzielonych na właściwy tekst artykułu i podsumowanie. Przedstawione wyniki prezentują się obiecująco, osiągając przyzwoite rezultaty pod względem standardowych metryk dla tego rodzaju zadań.

Biorąc pod uwagę przegląd literatury wykonany podczas realizacji pracy, jest to pierwsza próba wykorzystania technik abstrakcyjnego streszczania tekstów w języku polskim.

Contents

1	Introduction	6
1.1	Goal	7
1.2	Types of summaries	7
1.3	History	8
1.4	How humans summarize texts	9
2	Background	11
2.1	Abstractive text summarization	11
2.2	Text summarization in Polish	13
2.3	Automatic summaries quality	15
3	Data	16
3.1	Dataset preprocessing	17
3.1.1	Dataframe reduction	17
3.1.2	Cleaning	18
3.1.3	Trimming	20
3.2	Language utilities	20
3.3	Data splitting	21
3.4	Creating vocabulary	21
3.5	Summarization	22
4	Model	23
4.1	Sequence-to-Sequence	24
4.2	Encoder	24
4.3	Decoder	27
4.4	Attention	29
4.5	Model training flow	30
4.6	Model optimization	32
5	Training and experiments	34

6	Results	36
6.1	Not a real summary	36
6.2	ROUGE metric	36
6.3	Attention heatmaps	38
7	Conclusion	44
	Bibliography	46
	List of figures	52
	List of tables	53
A	Hadamard product	54
	Source code	55

Chapter 1

Introduction

Nowadays, the multitude of articles available online on every topic is overwhelming. In times when everyone has a very limited time, being up to date with all the interesting and important news is close to impossible. The initial notion behind this thesis was to explore possibilities to solve this issue. Potential solution is to create article summaries giving a general idea and key information. Such summaries are often available for scientific articles however, rarely accessible for other types of texts.

In news texts, summaries are mostly non-existent and people can rely only on knowledge from an article title or a lead section. However, they are not as reliable as a good summary, reflecting the article more precisely than a title or lead section. Titles are often created with an intention to catch readers' attention, not to describe an article's key information. Lead sections suffer from the placement of citation or author own thoughts, seldom being a good article information extractor. The benefit of using summaries is condensing the most significant information and providing an overall idea of an article. Having a possibility to create automatic article summaries allows for easier and faster information access, letting people survey a higher number of articles. Focusing on articles most appropriate with their expectations, they might become familiar faster with a given topic.

The essence of a good summary is conciseness, syntactical accuracy in a given language, and semantic coherence with the original text. A proper summary requires logical organisation and avoidance of redundancy. It needs to represent the most relevant information from the original article, despite being several times shorter. Summaries are often created specifically for a particular type of text. Each type requires a different level of generality and focus on details. We expect to get a different level of knowledge from scientific research text than from daily news. A valuable summary should include this information, to preserve a similar level to the original text. The information contained in a summary should be also relatively easy to comprehend for a particular type of reader. This is especially important, considering the requirement for condensing the key article information, for a summary to be still readable. In the beginning it might be

inconspicuous because for humans it feels natural to write understandable texts. Unfortunately, it becomes obvious when dealing with an automatic text summary, where the system needs to decide which piece of information to keep or withdraw. Some of the words, even if they do not give any relevant information, should be retained to preserve readability.

As an example, we could consider a man-made, well-prepared, and easily readable summary of a sample article. If we remove all non-informative function words and phrases, we produce a highly informational but incomprehensible text.

These are the attributes of a valuable summary whose creation is considered to be one of the most complex problems in the field of natural language processing. This field of science brings linguistic, information engineering, computer science, and data science together to work on topics related to understanding and analysing natural language. It requires not only natural language understanding but also natural language generation, to obtain state-of-the-art results.

1.1. Goal

The basic idea of the thesis was to help people stay up to date with new articles, considering the overwhelming growth of their number. The goal of this dissertation is to conduct experiments exploiting deep learning methods to obtain text summarization. This is a novelty, being the very first attempt of applying abstractive text summarization techniques for Polish language.

Work for this thesis involved creation of a software tool for automatic summarization. Due to a lack of previous work in this field specifically, a custom train and evaluation dataset, consisting of articles from websites in Polish language was created. The tool is designed using a set of powerful deep learning algorithms achieving very good results in tasks of text summarization in English. Project performance evaluation was held not merely using custom dataset, but also existing dataset [OK14] for extractive text summarization found during research.

1.2. Types of summaries

There are two main approaches for automatic text summarization. The first one is an *extraction-based summary* and its general idea is to create a summary by taking fragments of original texts. Each text portion could be a fraction of a sentence or whole sentence from the original text. The extracted content cannot be modified or altered under no circumstances. Such a summary should include phrases or sentences with text key information. The main challenge for this approach is to decide how to choose the most appropriate parts of the original text. A system generating extractive summary could work without text understanding, often relying on algorithms using bag-of-words techniques including words and phrases frequency.

An *abstraction-based summary* is a completely different approach to text summarization. In-

stead of relying on fragments of the original text, abstractive summary generates new sentences, that might not be present in the original text. This method builds a semantic representation of the original text and paraphrases it in new sentences. Abstraction transforms the extracted content with the help of semantic representation and context, compressing information from the original text. This frees the summary from the constraint of using only original text parts which results in generated summaries being more similar to human created ones. Additionally, an abstractive approach can support summarization in languages other than the original text.

The main drawbacks of the abstractive summaries are requirements for having a large training dataset and being computationally much more expensive than extraction. In new sentence generation it is also necessary to understand the domain of the original text, as it greatly improves the resulting summary. On the other hand, the extraction based summaries are less computationally demanding, do not require large datasets or domain understanding. Despite having low readability compared to the abstraction-based method, it still remains a more popular type of summary.

1.3. History

The introduction of automatic text summarization dates back to 1950s, starting with the paper *The Automatic Creation of Literature Abstracts* created by Hans Peter Luhn in 1958 [Luh58]. The author conducted research to create automatic abstract of technical papers and magazine articles. He used statistical information from word frequency and distribution to measure the significance of individual words and sentences. Text fragments scoring highest in significance were extracted to create "*auto-abstract*".

After general interest in creation of automatic abstracts, activity in the area declined for several years. Enthusiasm came back when artificial intelligence methods started gaining interest in the topic with the assumption that summarization requires text understanding [SP13]. This in turn required computing semantic representation of text for content identification. Hence it became used to measure the understanding capabilities of various systems. At that time however, the task was too complex which led interest in the area of text understanding to heavily decrease, and to reappear in recent years. Area of extractive text summarization enjoyed interest about a time when two evaluation programs appeared - Document Understanding Conferences (DUC) [ODH07] and Text Analysis Conferences (TAC) [DBL08]. Interest in abstractive text summarization gained interest very recently especially due to progress in computational speed using graphical processing units (GPU).

Early researches in text summarization were concentrated on single document summarization. In subsequent years, because of easier Internet access and exponential growth of information, problem of multi-document summarization gained attention. This approach considered

multiple documents related with each other by the same entity, topic, etc. Such summarization focused on extracting relevant information from each document while reducing redundancy or identifying contrary information. More recently, researchers conducted various studies on the topic of text summarization for languages other than English. The most popular systems included Japanese and Chinese languages but also a novelty focusing on multilingual approaches.

Since automatic text summarization gained attention, it was necessary to find good evaluation tools. Unfortunately it was difficult even for human generated summaries to define a common evaluation framework. For DUC [ODH07] and TAC [DBL08] manual evaluation protocols were introduced, even though they were highly subjective to judges. They measured several indicators to assess the quality of summarization. These were grammatical correctness, non-redundancy, focus on the most important information of the original text, structure, and coherence. In early 2000s, scientific articles focused on using *automatic evaluation* techniques. Common evaluation metrics include ROUGE [Lin04] and BLEU [PRWZ02], measuring the similarity between reference and generated summary. Their main drawback is that scores provided by those metrics, do not fully agree with human evaluation.

There still remain two major questions related to text summarization. How to select essential content of a document and how to express selected content in a condensed manner [SP13]? Research usually focused on the summary itself rather than on cognitive text understanding. This leads to the statement that the current limitations in various summarization algorithms would definitely benefit from better cognitive text understanding. This field was barely active in extractive text summarization, but gained significant attention in recent years in abstractive methods.

1.4. How humans summarize texts

Information about how human creates text summaries is not as obvious as it might seem. This topic is deeply integrated with psychology and cognitive science, concentrated on cognitive processes like attention, knowledge formation, reasoning, comprehension and language production. Expressive functions such as reading and writing are the outcomes of activity in multiple processes.

According to theories and cognitive psychology models, text coherence is very important for text comprehension [ZLG95]. Comprehension is an important field of research in cognitive psychology, because many researchers connect reading processes with activities in declarative, long-term human memory. Text comprehension associated with coherence is highly related to human cognition processes, which uses the existing knowledge and generates new knowledge.

To make text understandable, readers must establish coherent relations between informative parts of text [ZLLG16]. This is significantly important in a summary, where we usually want to

condense informative portions, preserving the overall level of text comprehension. Apart from text coherence it is also important to consider the level of text readability.

Creation of a summary clearly depends upon the type of original text. Reading a biography texts, we could skip less important facts like place of work or birth date and not experience loss in coherence. For narrative texts, which relies more on coherence, skipping a piece of text might reduce text understanding significantly. While considering a scientific article, we might skip a whole section of text and focus directly on chapters like abstract, results, and conclusions. Human creating a summary, should keep in mind the type of summarized text and target audience. This allows to determine the particular level of knowledge about the given area of topic and summary structure.

Chapter 2

Background

This dissertation is informed by several sources of related work. There are two major fields of research and we will start first with an overview of work on abstraction-based summarization. Next, we will focus on summarization in Polish language texts. In the end, we will concentrate on conclusions from research about assessing the quality of automatic summaries with comparison to manual summaries.

2.1. Abstractive text summarization

Methods applicable for translation are very similar to the ones used in abstractive text summarization. Due to this reason part of the publications cited below refers to the machine-translation topic.

The development in deep learning algorithms generated interest in textual data and started about the 1990s with the article proposing the *recurrent neural network* (RNN) [Elm90], allowing for processing data in temporal sequences. This class of neural networks was based on mechanism of *backpropagation* [RHW88], using hidden network units, separated from input and output data. Nowadays, recurrent neural networks have many applications in tasks with time dependent data like machine translation, speech recognition, time-series prediction, or proteins detection in bioinformatics. Unfortunately, despite being a meaningful breakthrough, they had several disadvantages, including problems with *vanishing gradient* [Hoc98] which tends to zero and *exploding gradient* which tends to infinity. Remedy to the first issue appeared in 1997 with the invention of *long short-term memory* (LSTM) [HS97] and in 2014 with *gated recurrent unit* (GRU) [CvMG⁺14]. Both were based on gating mechanism and prevented the vanishing gradient, but still suffered from the exploding gradient problem. The LSTM consisted of input, output and forget gates, regulating the flow of information through the cell. GRU mechanism was a simplified variant of LSTM, consisting of update and reset gates. Performance of both

mechanisms were similar and they are widely used in many state-of-the-art publications till today.

In recent years, research in abstractive text summarization methods took advantage from accelerated computation using graphical processing units and numerous available programming libraries supporting it. The advance started in 2010s with the introduction of neural-network-only-based approach for statistical *machine translation*, called neural machine translation. The idea was to solve the problem by encoding input sequence and extracting output sequence of variable length. The first work used *convolutional neural networks* (CNN), which often lost the ordering of the given words [KB13]. The next approach was to replace convolutions with GRU, encoding input sequence, one time step at a time and use another GRU to extract the output sequence [CvMG⁺14]. This approach was called *RNN encoder-decoder* architecture and gained interest in the field. The complex end-to-end solution for neural machine translation, called *Sequence-to-Sequence*, was introduced in the same 2014 year and was based on LSTM with limited vocabulary size [SVL14].

Despite the number of researchers working on neural machine translation and other sequential data related problems was high, the limitations of available methods were discovered. Even with LSTM and GRU mechanisms, architectures obtained poor performance on longer sequences [PABvM⁺14, CvMG⁺14]. The solution came from the *attention mechanism* [BCB15], providing additional information for encoder about the most relevant part of a sequence. Being a very powerful improvement, attention mechanism in different versions was largely adopted to many tasks and became a standard approach in many fields, including natural language processing and computer vision. Rush [RCW15] introduced neural attention Sequence-to-Sequence model with attention based encoder. Most of the abstractive text summarization tasks were handled on short documents. For long, multi-sentence summaries *hierarchical attention* mechanism was invented [NZdS⁺16]. Authors proposed a two-step attention mechanism, first on the word level to identify keywords in the document and second on the sentence level to identify key sentences, thus capturing the semantics, based on these two levels.

Results achieved with the attention mechanism were far beyond capabilities of pure encoder-decoder architecture. Some shortcomings like repetitions, insufficient information about the main idea, or grammatical problems, still remained. There were many novel approaches to solve issues such as applying convolutional gated unit at encoder output [LSMS18] or *pointer networks* [VFJ15] using the attention mechanism as a pointer to select the most probable input sequence as an output. This mechanism showed a great performance for abstractive text summarization [SLM17], being however partially an extractive approach. Some research based on attention went beyond using encoder and decoder completely. An influential article about *Transformer* architecture [VSP⁺17] depended only on feed-forward networks and attention mechanism. The architecture was based on convolutional gated unit and *Self-Attention* mechanism

drawing global dependencies between inputs and outputs.

Last few years were focused on rapid development in the field of natural language processing, especially in the area of translation and text summarization. The current state of the art methods are still not developed enough to fulfill all the human expectations of a good summary of a long text.

2.2. Text summarization in Polish

Advance in automatic text summarization for the English language encouraged researchers to tackle the problem for other languages. Experiments for Polish language resulted in many theoretical and working implementations, however, focusing on the extractive approach only. The reason behind it was the necessity to make a large dataset of texts with manually created summaries. In addition, text generation for the Polish language was much harder, as it was characterized by a high degree of inflection and complex syntax.

First research started around 2000s with the complex system *PolSum* [CG04] created in 2004, later evolving to *PolSum2*. System allowed for morphological, semantic, and syntactical analysis for extraction of the most important sentences. In addition it was able to recognize anaphora, increasing the coherence of selected sentences. At this time an independent article was published, analysing possibilities for further development of Polish text summarization techniques [BG05]. The authors optimized the solution for newspaper texts ranging from 10 to 50 sentences by calculating frequency of nouns, numerals and proper names for the text as a whole and for individual sentences.

Another text summarizing system *Lakon* was a result of development covered by the authors in several publications [Dud07, DSW08b, DSW08a]. The goal was to compare common existing extractive summarization methods for the Polish language. The system worked by selecting the specified number of sentences. The algorithm was based on sentence position in text assuming earlier sentences were more important and lexical chain, capturing semantic meaning of words and groups of similar words, in each sentence. It used also two algorithms to measure sentence importance, term frequency–inverse document frequency (tf-idf) [SM86] and Okapi BM25 [RWJ⁺94].

A few years later new publication – being a master thesis [Św10] – proposed the *Summarizer* system. It took advantage of a large number of machine learning algorithms on the level of sentence features. These features included:

- average log-likelihood ratio of words per sentence
- tf-idf score of words per sentence
- determination of sentence centrality, the average similarity to other sentences

- measuring of sentence similarity to the title
- existence of words indicating importance or non-importance according to the manually created lists
- position of a sentence in a paragraph and the whole text
- sentence, paragraph and text length
- number of capitalized words, numerals and punctuation
- sentence importance based on the first few words
- type of a sentence.

Based on the features, sentences with the highest scores were selected to create a summary. Finally, system was sorting the sentences in order of appearance in the original text, removing parts between parentheses and special forms at the beginning of sentences if such exists like ‘therefore’ etc.

Until 2014 there was no available dataset to analyse and compare new experiments in a field of automatic single-document text summarization for the Polish language. In this year the *Polish Summaries Corpus* [OK14] was introduced. It contained a large number of independent manual summaries of news articles. Each article had multiple summaries having 20%, 10%, and 5% of the original number of words. The corpus consisted of both extractive and abstractive summaries however, the number of 569 original articles and 10845 summaries in total was not enough to perform full training using deep learning methods. This dataset could be considered as a proper source of sample summaries for evaluation.

Although the next work on automatic text summarization adopted the usage of neural networks, they were introduced only for sentence selection [JP15]. Authors extracted sentences on a basis of various features on a sentence level, extending the list of proposed in *Summarizer* [Św10] by named entity appearance analysis. These included named entity types of person, organization, place, geography, date, and time.

Most recent studies were held by the same author to analyse the performance of extractive text summarization using co-reference information. The author’s idea benefited from using linguistic relation between two or more expressions, referring to the same object, like: *John eats. He was hungry*, where *John* and *He* refer to same entity. First experiments resulted in creating the *EMILY* system [Kop15], which was unable to outperform other systems for the Polish language extractive summarization [Dud07, Św10]. The first system evolved into what was introduced as *NICOLAS* [Kop19] and yielded promising results, outperforming other competing solutions. The author suggested the results could be improved, however, he also mentioned the possibilities of applying deep learning methods for text summarization.

2.3. Automatic summaries quality

Perfect automatic summaries should be considered as impossible to be distinguished from man-made, manual summaries. Quality of the summaries for English language was a subject of a study [OD11] about differences between human and automatic summaries. Authors analysed the performance of summarises of guided (related to a particular category) and not guided texts. Automatic summaries performed better on specific information, rather than vague topics. For news categories like accidents or attacks the key pieces of information were often mentioned directly at the beginning of text. In direct comparison, the density of information for automatic summaries was much lower than for manual ones. Content was often not compressed and included irrelevant, repeated information.

Overall automatic summaries returned at best less than half of all relevant information. Authors recommended to focus on guided summaries with known text categories as this gave a valuable and detailed diagnostic tool for better evaluation, because different types of text required different type of details preserved from the original text. In addition, the sentence position in original text was very important, often outperforming complex systems.

Chapter 3

Data

To avoid confusion, for the whole chapter 3 we will use the following naming convention. *Article* and *original text* refers to the content of a whole website article. *Text* or *source text* refers to the part of website article that we summarise. Finally, *summary* refers to the text being a summary of the *text* or *source text*.

The most significant part of research before starting the experiments was to find any existing Polish dataset suitable for the problem of text summarization. By ‘suitable’ we mean a dataset consisting of both a text to be summarised and a single or multiple assigned summaries. A few previous papers in the field of extractive text summarization in Polish language used custom created datasets. The only one, well-documented dataset is the *Polish Summaries Corpus* [OK14, CCD⁺14]. It was created by extraction of 569 articles from [Wei01] corpus. Each article was assigned with a number of independent manual summaries with 20%, 10%, and 5% length of source text. In addition, 154 articles were assigned with abstractive summaries. Unfortunately, total number of 569 original texts and 10845 summaries was insufficient for deep learning techniques.

Therefore creation of a completely new dataset was required. The design was inspired by datasets commonly used as a benchmark for English text summarization such as *CNN / Daily Mail* [HKG⁺15, NZdS⁺16]) and *XSum* [NCL18]. They have on average over 200,000 articles for training, with around 600 words for source text and 40 words for summary. As a result, the goal was to create a Polish dataset of over 200,000 source texts from popular Polish websites. The comparison between preprocessed dataset and other common datasets is presented in Table 3.1. The topics of websites were chosen to be spread into various categories like sport, business, news, technology, lifestyle and architecture. Articles were gathered using Scrapy tool [KL16] for fast development and efficient scraping. Due to the varying architectures of the websites and possibilities to explore them, the articles were not equally distributed among categories. Most of them were collected from sport and news-related websites. The origin, category, and number of articles per category are presented in Table 3.2.

Dataset name	Number of articles			Average length (in words)	
	Train	Valid	Test	Article	Summary
CNN	90,266	1,220	1,093	760.5	45.7
DailyMail	196,961	12,148	10,397	653.3	54.7
XSum	204,045	11,332	11,334	431.1	23.3
Current dataset	205,136 (total)			264.2	44.0

Table 3.1: Datasets comparison

Comparison between common datasets for abstractive text summarization: *CNN / Daily Mail* [HKG⁺15, NZdS⁺16] and *XSum* [NCL18] with Polish language dataset created for this project.

3.1. Dataset preprocessing

Each article in the dataset included columns with a url, title, lead part and paragraphs. Some comprised also bullet-point lists inside or between the paragraphs. During preprocessing step, the dataset was reduced, cleaned and trimmed to match the desired level of data consistency. Prepared title and lead part columns served as a summary while merged paragraphs column became source text for summarization.

3.1.1. Dataframe reduction

The very first set of operations performed on a raw dataset was a reduction in the number of existing columns. The dataset contained rows like a url which were redundant and should be removed.

Raw source texts and summaries taken from websites may have appeared in a form of multi-sentence strings or lists of sentences. To standardize format, all texts were transformed into lists of sentences.

During data exploration, it turned out lead section sentences may have been repeated at the beginning of paragraphs section. Because of the way summary and article parts were generated, duplicates could appear in both parts. To remedy this problem, existing duplicated sentences were removed from the paragraphs and remained unchanged in lead sections.

In order to run experiments, model required exclusively source text and reference summary. To obtain such format, title, and article lead section were merged into a summary column. Other columns with article paragraphs and bullet-points were merged into a source text column.

Finally, all rows containing empty source text or summary were removed, reducing the size of the dataset.

Website		Number of articles		
Link	Category	Raw	Cleaned	Trimmed
krakow.naszemiasto.pl	news	356	274	260
krakowwpigulce.pl	news	5,905	5,839	4,501
gazetakrakowska.pl	news	12,946	10,719	9,684
nowiny24.pl	news	67,651	55,253	52,902
polki.pl	news, lifestyle	25,686	17,089	15,521
sport.pl	sport	78,339	73,628	71,911
bankier.pl/moto	technology	797	774	677
nowymarketing.pl	technology	5,280	5,269	2,422
wirtualnemediia.pl	technology, business	6,664	6,664	6,657
spidersweb.pl	technology, lifestyle	29,133	28,763	28,665
mamstartup.pl	business	93	93	50
wnp.pl	business	476	476	331
strefainwestorow.pl	business	3,536	3,524	3,509
bankier.pl	business	3,809	3,803	3,327
sztuka-architektury.pl	architecture	3,939	2,177	1,108
bryla.pl	architecture	4,702	3,907	3,611
Total		249,362	218,252	205,136

Table 3.2: Dataset sizes

Raw articles refer to raw articles taken directly from the websites. Cleaned articles refer to documents with non-empty source text and summary, with removed html signs, special characters and hidden numbers. Trimmed articles refer to articles with number of words above the minimal threshold and trimmed to the maximum desired length.

3.1.2. Cleaning

The next set of operations was focused on cleaning the dataset from various non-standard characters. At the beginning, all the existing html elements and different encoding characters were removed. Signs appearing in source texts and summaries which were not required for summarization were removed or replaced. It should be mentioned here that by deleting characters, we also removed additional information contained in the text. For example, a quotation mark could suggest a named entity, brackets could contain a separate nested sentences or an explanation for citations. Nevertheless, we discarded these characters, to reduce the multitude of unique words in dataset, decreasing computation time, and memory demands. Table 3.3 shows the detailed list of patterns deleted from the dataset. Because of the manner how white spaces were removed, output sentences had a single space between the last word and the dot.

In addition to single signs removal, cleaning process included replacement of numerical characters. To reduce the number of possible unique numbers in source texts and summaries, all numbers, date and time formats were replaced by a special token *<num>*. For future experi-

Pattern	Replacement
\t, \r, \n	single space
text within brackets and braces {} or []	nothing (empty)
_, +, -, , *, /, .., :, ,	single space
<, >, (,), &, ©, ø, , (comma), ;, \, ,, ", " ", ”, “, ’, ‘, ’, ‘, - , - , — , — , %, ^, \$, #, @	nothing (empty)
?, !	. (dot)
white spaces	single space

Table 3.3: Dataset cleaning rules

Removing unnecessary signs from dataset. Pattern column shows characters replaced with corresponding replacement.

ments is will be worthwhile to prepare several categories of entities such as number, date, time, age, or money. This operation will allow for better representation of the context of a text than while using a plain *<num>* token for every occurrence of a numerical character. In this work, we considered all types of numerical characters as a single token. As a result, we dismissed significant pieces of information, important, especially in business articles. For example, we no longer have a possibility to distinguish between the amount of currency and the date of the bank transaction. Though, the main goal at this stage of the project was the reduction of memory requirements and the time required for training. Details are presented in Table 3.4.

Numerical characters	Replacement
unlimited number of numerals 0-9	<i><num></i>
floating point numbers formats 0.50, 15,25	<i><num></i>
date formats 02\02\1999, 05/05/2005, 31.12.2000, 20-08-2008	<i><num></i>
date formats with Roman numerals for month 31.XII.2000	<i><num></i>
time formats 06.30 11:45	<i><num></i>

Table 3.4: Numerical types replacement

Words in source texts and summaries appeared in many inflected forms. To limit the number of unique words, we decided to group the inflected forms of words together. In general, there exist two possible approaches. An easier technique was to use *stemming* that chops off the end of a word, resulting in the core base of a word, called a *stem*. Stemming operates on single word at a time, without considering context and as a results it cannot discriminate between different parts of speech having the same word form. Due to these drawbacks, second approach called *lemmatization* was used. This process takes into account morphological analysis of words and obtains the common base form called a *lemma* or a dictionary form. This is especially important in Polish which is a strongly inflected language. For example, having a word *pies* meaning *dog*, it could appear in documents in such forms: *pies, psa, psu, psem, psie, psy, psów, psami, psom, psach*. Applying lemmatization to all those forms results in single *pies*

(*eng. dog*). This helped significantly reduce the number of unique words, which was important for a model and made documents easier to understand. The tool used for lemmatization was a Polish language model implementation for spaCy [TK19, PPR19], based on Polish system *Morfeusz* [Wol14, CCD⁺14]. (More on this tool in section 3.2. *Language utilities*.)

Additionally, because the lemmatization process had a high overhead it was parallelized, allowing computation time reduction from 7h24min to 2h20min.

From this point forward, every occurrence of a *word* or *words* refers to *lemma*. Every occurrence of *text*, *source text* or *summary* refers to text after lemmatization.

3.1.3. Trimming

Length of cleaned source texts and summaries varied significantly. To decrease the dispersion, documents below minimum length threshold were removed and the remained were trimmed to maximum length threshold. The minimum length limit for source text and summary were set to 40 and 10 words, respectively. The maximum length limits were set to 400 and 100, preserving the ratio around 25% between summary and text. What is also important, trimming was done at the end of the sentence, not in the middle of it. This is the reason why both source text and summary are mostly shorter than threshold. Similarly to lemmatization trimming process was parallelized, reducing time from 7h32min to 2h35min.

3.2. Language utilities

To conduct experiments, a set of utilities for working with spaCy [TK19, PPR19] language model was created. The core functionality of module was setting up a spaCy language model to support Polish language, enabling to perform operations such as lemmatization or *tokenization*. Both were crucial for source texts and summaries preparation for training. First operation, lemmatization was described earlier in 3.1.2. *Cleaning* section. The role of tokenization is to segment text into words and punctuation marks called tokens. This process was used to separate sentences into single words.

The next utility included a series of special tokens to work with textual data. The first pair of tokens marking the beginning and end of a document, *<sos>*, *<eos>* were created (short for *start of sequence* and *end of sequence*, respectively). All source texts and summaries were surrounded by these tokens on each side. Because some words appeared only in the test part of the dataset, it was needed to create a special *unknown* *<unk>* token which was used every time the word was not present in the vocabulary. To reduce the level of memory usage, the dataset was fed into the neural network in batches. Loading dataset samples in a form of batches, reduced the number of loss values to compute at once during backpropagation process, increasing the speed of training. Because the length of source texts and summaries can be different on a batch level,

we introduced padding token `<pad>` responsible for replacing missing words in a batch. The last custom token was a number token `<num>` used in 3.1.2. *Cleaning*. This token represented integers, floats, dates and time in different formats as a single token as shown in Table 3.4 above, significantly reducing the unique number of words.

Neural network model was unable to work directly on textual data, as it was required to calculate its performance at every step. To meet the requirements, texts were transformed into lists of numbers. The transformation between words and numbers (indices) was also a feature of a language utilities. It supported two main operations, *text to indices* and *indices to text*, first responsible for encoding words into numbers and second for decoding numbers into words.

3.3. Data splitting

Complex neural networks models such as the one created for these experiments, were prone to fit too closely to limited data and perform poorly on new, unseen data. Such a situation is called *overfitting* and to monitor it the dataset was split into training, validation, and testing part. Pairs of source texts and associated summaries were divided with ratio of 0.8, 0.1, and 0.1. For all 205,136 pairs, it resulted in 164,108, 20,514, and 20,514 samples respectively.

3.4. Creating vocabulary

The neural network required to know the size of an input, which depended directly on the vocabulary and its size. Vocabulary consisted of all unique words in a dataset. Without cleaning of unnecessary characters, lemmatization, and trimming, over 200,000 raw source texts and summaries contained 1,416,100 unique words. After all preprocessing steps the number of unique words was equal to 415,121. Unfortunately, over 3-fold reduction in vocabulary size was not enough to perform training.

To further reduce the number of words, we decided to choose only words that appeared at least 10 times in the whole dataset. The word is included in a dictionary even if it appeared only in a single article, but not less than 10 times. The obvious drawback of this solution is discarding key information such as a surname or name of a place that appears only in a single text and less than 10 times. An additional possible risk was the inclusion of misspelled words, which were not removed during the previous cleaning stages. Finally, we managed to lower the number of unique words to 68,725, including unfortunately 309 misspelled words, which were skipped during the cleaning process. Miswritten or words with typos were detected by manual analysis of the whole vocabulary. The analysis of wrong words was conducted after model training, meaning these words could appear in output summary. This 6-fold reduction resulted in a decrease in the learnable parameters of the model by over 321M to 55M.

3.5. Summarization

Text summarization is the output of model prediction. Data for summarization was intended to be a single or multi-sentence, single string. The source text was cleaned and lemmatized using the same algorithms as described in 3.1.2. *Cleaning* section. Length of source for prediction procedure was however not trimmed. To match the same level of unification as data available during training, source text was surrounded by `<sos>` and `<eos>` tokens. Such prepared source text was filled directly into the neural network model. The goal was to receive a summary, with a length up to 25% of the original source text or until neural network generated ending `<eos>` token.

Chapter 4

Model

The architecture of the neural network model used in experiments was a Sequence-to-Sequence (*Seq2Seq*) mechanism [SVL14] in which the eponymous sequences corresponded to encoder and decoder GRU [CvMG⁺14] parts. To increase capability, architecture was enhanced by applying Bahdanau attention [BCB15] mechanism. These core building blocks constitute a widely used and powerful model for tasks including machine translation or speech recognition.

Experiments code was written in Python3 with additional libraries. The model itself was created using PyTorch library [PGM⁺19], supporting Nvidia CUDA technology [Kir07] for accelerated computation using GPU. Language model and utilities function were created using spaCy [TK19, PPR19].

Prior to feeding the encoder with bare texts, data was cleaned and transformed into sequences of numbers, as described in previous sections 3.1. *Dataset preprocessing* and 3.2. *Language utilities*.

The number of model learnable parameters depended directly on the number of unique words in the vocabulary. As mentioned in 3.4. *Creating vocabulary*, vocabulary consisted of words with at least 10 occurrences in the whole dataset. Table 4.1 presents a comparison between the number of model parameters in the case of using every existing unique word and only words with at least 10 occurrences in source texts and summaries.

Model part	Every unique word	Words with at least 10 occurrences
Encoder	54,911,104	10,572,416
Decoder	266,717,201	44,677,365
Total	321,628,305	55,249,781

Table 4.1: Model learnable parameters comparison.

Encoder, decoder and total number of parameters while using every word from dataset and including the ones appearing at least ten times.

4.1. Sequence-to-Sequence

Sequence-to-Sequence model took as input sequences and output another sequences. Following the original articles [CvMG⁺14, SVL14], architecture consisted of two RNN blocks called encoder and decoder, and managed the flow of data during training and evaluation. The encoder block was responsible for encoding the complete information about input representation using RNN into single fixed size vector called *context* or *hidden state*. This state was sent into decoder, which also consisted of RNN and generated the output sequence. Because the output sequence was dependent only on a single context vector, standard Seq2Seq architecture was not capable to perform well on long sequences. To overcome this issue, in experiments we enhanced decoder with the attention layer by adding more information for decoder. This helped to choose the most appropriate word for an output.

Details about encoder and decoder are described in the following sections 4.2. *Encoder* and 4.3. *Decoder*, while attention mechanism is presented in 4.4. *Attention*.

```
Seq2Seq(  
  (encoder): EncoderRNN(  
    (embedding): Embedding(68725, 128, padding_idx=1)  
    (gru): GRU(128, 256, num_layers=2, dropout=0.5, bidirectional=True)  
  )  
  (decoder): DecoderRNN(  
    (embedding): Embedding(68725, 128, padding_idx=1)  
    (dropout): Dropout(p=0.5, inplace=True)  
    (attention): BahdanauAttention(  
      (attention): Linear(in_features=512, out_features=256, bias=True)  
    )  
    (gru): GRU(384, 256, dropout=0.5)  
    (classifier): Linear(in_features=512, out_features=68725, bias=True)  
  )  
)
```

Figure 4.1: Model architecture overview

4.2. Encoder

Encoder's objective was to transform sequences of input into *embeddings*, obtaining the lower-dimensional representation. Embedded input then fed into a type of recurrent neural network, a bidirectional GRU [CvMG⁺14], which stored the hidden state of input sequences. The final hidden state of the encoder was called a *context vector* and was pushed as an input for the decoder.

Embedding layer Storing words as a list of 68,725 indices was not efficient. To reduce model parameter number, we applied dimensionality reduction: embedding. This is a mapping of categorical variable, a discrete number of indices into continuous number. Embedding could be

understood as an array of learnable weights, transforming from indices into vectors. In this work, embedding was represented as an additional, trainable layer located at the beginning of encoder model. Such format helped create a representation that better mimicked the true similarity between words. In this model embedding reduced size from 68,725 into 128. As an example, Polish words *mecz* (eng. *match*) and *boisko* (eng. *football pitch*) should be close to each other at the output of embedding layer, suggesting strong correlation between words in texts.

RNN The core of encoder was a recurrent neural network. Standard feedforward neural networks propagates the input in a single direction, from the first to the last layer. Information about individual input does not depend on any particular previous input, which is enough for image data (for which typically convolutional neural networks, CNN, are used). The convolutional neural network focuses on spatial information from a current input image only, convolving it using filters and extracting a feature map. For text however, the information about a previous word is crucial. Recurrent neural networks are suitable especially for this type of data, because they hold information about previous states. A diagram of a simple RNN is shown in Figure 4.2.

Unfortunately, standard recurrent neural networks have several disadvantages. Most significant is poor performance in storing long dependencies. The same weights influence an output y_t during all the time steps in training. During the backpropagation process, values of gradients change from the very end to the beginning of a computational neural network graph. In the next step, the calculated gradients alter neural network weights, responsible for correct output for the whole text sequence. For long sequences, values of gradients might drop significantly, resulting in loss of change in neural network weights, called the vanishing gradient problem [Hoc98]. This reduces the accuracy remarkably and to mitigate these drawbacks, improved architectures were introduced like GRU [CvMG⁺14], described in the next paragraph.

Figure 4.2: Recurrent neural network (RNN)

$x_1, x_2, \dots, x_t$ represent input words from a sequence.

$y_1, y_2, \dots, y_t$ represent output (predicted) next words.

$h_0, h_1, h_2, \dots, h_t$ represent RNN hidden states from previous input.

Initial state h_0 could be set to *null* or an arbitrarily chosen value.

Gated Recurrent Unit (GRU) [CvMG⁺14] is a type of recurrent neural network used in this project. Main purpose of this architecture is to overcome the aforementioned vanishing gradient problem [Hoc98], keeping information from previous sequences longer than standard RNN. GRU overcomes short-term memory issue using update gate and reset gates, responsible for regulating what information to keep or discard at every time step, and hidden state. Gates can be interpreted as vectors, containing data ranging between 0 and 1, which are multiplied by input and hidden state. Diagram representing GRU architecture is shown in Figure 4.3.

In this project we used hidden state size of 256 and bidirectional version of GRU, enabling two-way text analysis. It helped to preserve context in long texts and provided additional information for the beginning of a sequence that normally would appear in the last time steps.

Figure 4.3: Gated Recurrent Unit (GRU)

GRU is built in the following way. Update gate is responsible for how much information from the previous time steps to carry over to the current state. It is calculated by multiplying previous hidden state h_{t-1} and current input x_t by the corresponding weights, summing them and applying sigmoid function (4.1), resulting in values between 0 and 1. Equation for update gate is shown in equation (4.2).

$$S(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1}, \quad (4.1)$$

$$z_t = \sigma(W_z * x_t + U_z * h_{t-1} + b_z) \quad (4.2)$$

Reset gate is very similar to update gate, but its purpose is to determine value of past information to forget. The equation looks almost the same, but weights for previous hidden state h_{t-1} and current input x_t are different. Equation for reset gate is show in equation (4.3).

$$r_t = \sigma(W_r * x_t + U_r * h_{t-1} + b_r) \quad (4.3)$$

A candidate hidden state h_t , denoted $\tilde{h}_t$, decides how much of previous information to keep, comparing to current input. We start by multiplying the current input x_t and the previous hidden state h_{t-1} by their weights. Next, we apply the element-wise multiplication between reset gate r_t and output of multiplication h_{t-1} and weights from previous step. This element-wise multiplication $\odot$ is known as *Hadamard product*, which differs from matrix multiplication $*$ and is described in Appendix A. At the end we sum both values and apply *hyperbolic tangent* function ($\tanh$) (4.4). The formula for hidden state candidate is shown in equation (4.5).

$$\theta(x) \equiv \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}, \quad (4.4)$$

$$\tilde{h}_t = \theta(W_h * x_t + U_h * (r_t \odot h_{t-1}) + b_h) \quad (4.5)$$

To obtain final hidden state h_t for time step t , we apply element-wise multiplication twice: the first one between update gate z_t and previous hidden state h_{t-1} , and the second one between $1 - z_t$ and candidate for hidden state $\tilde{h}_t$. The former is responsible for keeping a part of previous hidden state, the later for storing a candidate. Finally, both parts should be summed up to produce the final hidden state h_t , which is shown in equation (4.6).

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t \quad (4.6)$$

To improve model performance, the bidirectional GRU version was used in the model. By using it, we should imagine having two sequences, in standard and reversed order. Visualization of bidirectional recurrent neural network is shown in Figure 4.4.

4.3. Decoder

Decoder's role was to take the context vector (the last hidden state of encoder), as an input for the first recurrent layer. Similarly to encoder, decoder also consisted of embedding layer, reducing dimension from 68,725 to 128 and using GRU with a hidden state size of 256. Contrary to the encoder, this GRU layer was not bidirectional. After embedding layer, we used regularization mechanism called *dropout layer* [HSK⁺12], which ignored particular neural network units and helped prevent model from overfitting. In essence by applying dropout we zeroed the output of the fraction of neurons from previous layer. It allowed the model to focus more on particular, important features. Moreover it reduced probability to adapt only to training data and performed poorly on testing data. In evaluation mode, dropout layer was disabled, to preserve result immutability. The visualization of dropout is shown in Figure 4.5.

Figure 4.4: Bidirectional recurrent neural network

Figure 4.5: Dropout layer visualization

Right after dropout layer, we placed attention mechanism, details of which are described in the 4.4. *Attention* section below. At the very end of decoder, there was a linear classifier of size 512, to decide what output should be produced.

Comparing to encoder which worked always the same way, decoder had two different modes: *inference*, also called *generation* process, and *training* process. During the training phase, the goal of the model was to predict the next word of a target sequence, given a previous word of target sequence. In generation phase, model passed the last predicted word as an input for the next time step. For the purposes of this project, during training phase we used mechanism called *teacher forcing* [BVJS15], which with a probability of 0.5 selected between taking the previous word from target sequence and the last predicted word. This allowed the model to converge quicker to the better results.

To explain how this mechanism works, we might think about a situation that – especially at the beginning – the model will mostly output wrong sequences. Because we use recurrent neural network, next predicted words depends on the previously predicted one. If our model makes a mistake in the fourth word in a 200 words long sequence, it is very likely all the words from the fifth till the end will be wrong. It might happen, even if all the calculations apart from this particular one are correct. Using this technique, we need to remember about lack of such ground truth target for test or summarization task.

4.4. Attention

In standard Seq2Seq model only the context vector is responsible for passing complete information about sequence from encoder to decoder. A single vector cannot store enough relevant information, required for longer sequences. To overcome this problem, the attention mechanism was introduced. It works by focusing on particular words in a sequence using *attention weights*. At every time step, apart from encoder context vector, decoder is served with the information to which part of the sequence it should give more attention. Intuitively attention could be understood as a weighted vector and our goal is to learn values of those weights.

The attention mechanism used here is called *Bahdananu attention* [BCB15]. It used all encoder hidden states and decoder last output instead of just encoder context vector.

The attention mechanism could be explained in the following steps. The first one is to get all the encoder hidden states, instead of only final hidden state. During the next step, we calculated the *alignment scores* between each encoder output and decoder input. We multiply encoder outputs and decoder hidden state by their own weights and sum them up into attention linear layer. In our experiments, the size was set to 512 neurons. Then we apply hyperbolic tangent function, equation (4.4) and apply matrix multiplication with trainable

vector of attention layer inner parameters. The whole formula is presented in the equation (4.7).

$$e_{ij} = W_{all} * \theta(W_{dec} * H_{dec} + W_{enc} * H_{enc}) \quad (4.7)$$

Alignment score e_{ij} scores - how well inputs around position j and the output at i position match to each other.

Having the alignment scores, we calculate attention weights by applying *softmax* function, as given below:

$$a_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^K \exp(e_{ik})}, \quad (4.8)$$

where a_{ij} are attention weights of j -th source word and i -th target word. As a result all weights are spread in range between 0 and 1, with all together summing up to 1.

To obtain the context vector for attention, we multiplied encoder outputs with attention weights, as shown below

$$C_i = \sum_{j=1} a_{ij} * h_j, \quad (4.9)$$

where the context vector c_i depends on the sequence of annotations $h_1, \dots, h_T$ to which the encoder maps the input sequence.

The final step of attention is to predict the output word. We feed the GRU mechanism with the result of concatenating context vector and the previous decoder output. The output of GRU is used in the next time step as new hidden state. The final output of decoder is generated by pushing it through a linear classifier, setting the probability scores for next predicted word. Visualization of Bahdanau attention mechanism is shown in Figure 4.6

4.5. Model training flow

The flow of data in training the Sequence-to-Sequence model was organised in a following way. We started by passing a batch of data into encoder. Inside encoder we pushed sequences through embedding layer, reducing dimensionality from 68,725 original words into 128. Next we fed GRU mechanism, having 256 features in the hidden state with the embedded sequences. As the last step of encoder, we merged the output features from the bidirectional GRU in order to supply the decoder.

Encoder returned merged GRU output features together with all its hidden states. The following step was to initialize decoder hidden state, at the beginning with the hidden state of encoder. Likewise we initialized decoder output token with $\langle sos \rangle$ token and also created an empty vector for a model generated summary. The next step was to create a loop within the range of maximum allowed summary sequence length. Here we used the reference summary length for each batch. Inside this loop we fed the decoder with the previous decoder output,

Figure 4.6: Bahdanau attention visualization

decoder hidden state, and the encoder output. In the first iteration, decoder hidden state was represented by encoder hidden state and decoder output was a `<sos>` token.

Inside the decoder we started similarly as within encoder by pushing the previous decoder output through embedding layer and reducing the size from 68,725 to 128 features. Then we applied dropout layer on neurons of neural network with the probability of 10% on the embedding layer output. At the same time we invoked attention layer with encoder output and hidden state. We concatenated them and used fully connected layer with 512 neurons to calculate alignment scores. After this step, attention mechanism enabled us to apply softmax function on the alignment scores and to determine the attention weights, which were returned back to the decoder. Inside decoder we calculated context vector using the encoder output and attention weights obtained in the previous step. Then we concatenated context vector with embedded previous decoder output and fed the decoder GRU mechanism with 256 features in hidden state. Recurrent GRU unit yielded the output features and hidden state. As the final decoder step we applied fully connected linear classifier with 512 neurons input and 68,725 output features on the recurrent neural network output features, obtaining new decoder output.

Values returned by the decoder were the new decoder output, new GRU hidden state, and attention scores. New decoder output updated the vector with summary tokens. The previous decoder hidden state got updated with the newly obtained hidden state. Because we used teacher

forcing mechanism [BVJS15], the previous decoder output was updated with a probability of 50% between the new decoder output or token from the reference summary at particular step. The loop ended when the maximum allowed summary length is reached and the generated summary vector is returned.

4.6. Model optimization

Adam optimizer Choice of an optimization algorithm for a model has a huge impact on performance and convergence time. The algorithm used in this project was the Adaptive Moment Estimation - *Adam* optimizer [KB14]. Adam is an adaptive learning method which computes learning rates for every single parameter. Moreover it uses the mean (first moment) of gradient m_t and uncentered variance (second moment) v_t . First hyperparameter for this optimizer is α , responsible for learning rate. Next one are β_1 and β_2 that are close to zero and controls decay rates of moving averages. The last one is a small epsilon which prevents from zero division issues. In experiments we specified learning rate to 0.001 and β_1 and β_2 to 0.9 and 0.999, respectively. The equations for calculating Adam optimizer are shown in the equations (4.10, 4.11, 4.12).

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad (4.10)$$

First moment or mean m_t .

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \quad (4.11)$$

Second moment or uncentered variance m_t .

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \quad (4.12)$$

Adam optimizer update rule.

Cross Entropy Prediction of the next word for summary requires taking the best matching words. This problem is called a multi-class classification, where different classes are represented by words in vocabulary. Commonly used loss function for such type of task is called *cross entropy*. It allows for fast training and good level of generalization. It measures the difference between two probability distributions: P, a known probability of each class label and Q, model predicted probability of each class label. Prior to calculating cross entropy, it is needed to obtain predicted word probability by applying softmax function equation (4.8). As a result, having a target word from reference summary and predicted probability of each word, we can calculate

the value of loss function. Formula for calculation of cross entropy loss is shown in the equation (4.13).

$$CE = -\log \left(\frac{e^{q_p}}{\sum_j^C e^{q_j}} \right) \quad (4.13)$$

C - class labels, q - model output, p - the only one positive class

Gradient clipping Gradient explosion and vanishing are problems often occurring in recurrent neural network. This model got significant number of parameters and during initial training trials it crashed due to gradient explosion. To eliminate this issue the model used *gradient clipping*, which set a value threshold for all the gradients. Besides precluding gradient explosion it stabilizes the learning process and allows for training acceleration, especially in language tasks [ZHSJ20].

Chapter 5

Training and experiments

Time consideration Typically training a deep neural network involves thousands of short-lasting epochs required for network to converge and get satisfactory results. Due to the large number of articles in dataset and length of those articles, in this experiments every epoch was computationally expensive. As an example a single epoch of training using Nvidia Tesla P40 GPU took on average 5 hours or 2.5 hours using Nvidia Tesla V100. As a result we performed only 40 epochs of training, followed by validation on separate part of dataset.

Memory consideration The size of batch was restricted to just 16, due to GPU memory limitations. With such a small batch, the model did not exceed 14GB memory requirement at its peak, with longest sequences, mostly using around 12GB. Despite available GPU had more than double of model maximum required memory, restricting model not to surpass 16GB, allowed for later fine-tuning on platforms like Google Colab, Kaggle or to use a smaller capacity GPUs. For comparison, In the inference mode during prediction, model required around 6GB of memory. Total number of source texts and summaries pairs for training was 164,108. With batch size of 16 it indicated 10,257 batch iterations per epoch. Validation and test datasets both had 20,514 articles, translating to 1,283 batch iterations. Each saved model weights requires up to 227MB free space disk. Weights were saved in a manner of allowing to fine-tune them using both GPU and CPU.

Model training In experiments, model was not trained by a single training procedure execution. After each epoch model weights were saved and got ready for weights fine-tuning. Final model was trained in 4 major steps, 10 training epochs per each step. For each next step, model was loaded with the weights saved at the end of the previous step. Reason for this was that after 10 epochs, value of value of train cross entropy loss function, equation (4.13), stopped improving. However, the small reduction was visible for validation loss. Importantly training loss was significantly lower than validation loss due to teacher forcing mechanism [BVJS15], which was not used for evaluation.

During training we tried mechanism of stepping down learning rate for Adam optimizer. The starting learning rate was set to 0.001 and it was dropped by 25% after every epoch. Analysing the model loss it was clearly visible that after about 8th learning rate drop, when it reached 0.0001 value the training almost disappeared. As a result, we decided to start with learning rate of 0.001 for each next 10 epoch training, after loading model trained weights.

Prediction Prediction process for sequence to sequence model was different than training. We did not use teacher forcing mechanism as we did not have a reference summary text to base on. In case model was fed with the sequence having word unknown for the model, they were changed into `<unk>` token. The goal of prediction process was to receive a summary of a given source text. Model generated summary had the length of up to 25% of the source text or until the time step at which the `<eos>` token was generated. This terminated the summary generation and yielded the output summary.

Chapter 6

Results

6.1. Not a real summary

Before analysing the results, we need to clarify the crucial problem related to the created dataset. Common datasets for summarization task always used human generated or assisted reference summaries for original source texts. Because of that, models and solutions generated in other articles always assumed the correctness of these summaries. Unfortunately we were not able to confirm the same assumption for this dataset.

We were not able to validate all samples in our data. As a results, there may have been errors in the process of obtaining articles. Most of errors were eliminated during data preprocessing, however there might appear other issues. Reference summary part was generated from an article title and header or lead part of website article. Source text for summary was generated using all the other website article parts gathered from an article. Due to varying architectures of websites there might appear examples where original text is not fully related to reference summary. Such a situation could happen when on-site advertising hid part of the article or was integrated with the text. On the other hand, article title could have been written to attract customers and tended to poorly summarize text. Fortunately, there was only a little chance to obtain such a bad summary, not related to original text.

In this situation, it is important not to take for granted the results from metrics.

6.2. ROUGE metric

As a qualitative measure of results we have chosen a set of *ROUGE* metrics [Lin04] which stands for Recall-Oriented Understudy for Gisting Evaluation. This is a set of metrics for evaluation commonly used in machine translation and text summarization tasks. It compares model output with reference target. We are focusing on three accuracy measures: recall, precision and F-score. In case of ROUGE, recall stands for number of overlapping words, divided by number

of words in reference summary, equation (6.1)

$$recall = \frac{number_of_overlapping_words}{number_of_words_in_reference_summary} \quad (6.1)$$

Precision refers to number of overlapping words, divided by number of words in model generated summary, equation (6.2)

$$precision = \frac{number_of_overlapping_words}{number_of_words_in_generated_summary} \quad (6.2)$$

For better overview of a model performance use F-score, calculated from both precision and recall values, equation (6.3)

$$F_score = 2 * \frac{precision * recall}{precision + recall} \quad (6.3)$$

From a whole family of ROUGE metrics we decided to use ROUGE-N and ROUGE-L. The first one measures the number of overlapping n-grams between the reference and model generated summary. We measured ROUGE-N for unigrams and bigrams, obtaining ROUGE-1 and ROUGE-2 values respectively. ROUGE-L measures longest matching sequence of words using longest common subsequence (LCS). It operates on a sentence level, by identifying the longest common n-grams between both summaries.

Comparison of ROUGE metrics with previous experiments is hard because of custom created dataset. Considering this fact we have measured metrics values for test data, not available for a model during training. Additionally, we tested our model on The Polish Summaries Corpus [OK14] dataset both without fine-tuning and with fine-tuning for a single epoch. In both cases, we used vocabulary taken from the custom dataset, which might impact the results. For comparison, the used vocabulary consisted of 68,725 unique words while The Polish Summaries Corpus consisted of 19,814 unique words, cleaned and lemmatized using the same algorithms as described in 3.1. *Dataset preprocessing*. Both vocabularies got only 15,387 overlapping words. The results are available in Table 6.1. Additionally, we provide a comparison between our results and set of extractive-based approached, available in Table 6.2.

The performance of our model was benchmarked with other trials of English language abstractive text summarization results [RCW15, NZdS⁺16]. We used the same ROUGE F-score metrics and obtained results that were about half off from the benchmarks. These results confirmed the correctness of our model, indicating a wide possibility of further improvements. The summary is shown in Table 6.3.

	ROUGE-1			ROUGE-2			ROUGE-L		
	Recall	Precision	F-Score	Recall	Precision	F-Score	Recall	Precision	F-Score
Polish Summaries Corpus	0.07	0.12	0.10	0.01	0.01	0.01	0.06	0.49	0.10
Polish Summaries Corpus with fine-tuning	0.15	0.15	0.15	0.03	0.03	0.03	0.13	0.68	0.21
Custom dataset	0.15	0.17	0.15	0.04	0.04	0.04	0.13	0.53	0.16

Table 6.1: ROUGE-1, ROUGE-2, ROUGE-L scores comparison on test set.

ROUGE metrics scores comparison between custom and The Polish Summaries Corpus [OK14] datasets. Values are measured on test part of a data, not available during model training. The metric comparison is not fully proper, as described in 6.1. *Not a real summary*.

Recall on Polish Summaries Corpus		
System	ROUGE-1	ROUGE-2
Our	0.07	0.01
Our with fine-tuning	0.15	0.03
Baseline [OK14]	0.56	0.21
Świetlicka [Św10, OK14]	0.58	0.23
TextRank [OK14]	0.57	0.19
Nicolas [OK14]	0.59	0.24

Table 6.2: Comparison with extraction-based methods on Polish Summaries Corpus.

ROUGE-1 and ROUGE-2 metrics recall comparison on The Polish Summaries Corpus [OK14] dataset with extraction-based different systems were presented in last year paper [Kop19]. The significant difference in metrics value is related to ROUGE being an imprecise metric.

ROUGE metrics F-Score in different trials			
	ROUGE-1	ROUGE-2	ROUGE-L
Custom dataset	0.15	0.04	0.16
Rush et. al 2015 [RCW15] on DUC [ODH07] dataset	0.22	0.06	0.19
Nallapati et. al 2016 [NZdS ⁺ 16] on DUC [ODH07] dataset	0.27	0.08	0.23

Table 6.3: ROUGE metrics F_1 scores trials comparison.

Comparison of F_1 scores for ROUGE-1, ROUGE-2 and ROUGE-L metrics between our model on custom dataset and two models on DUC [ODH07] dataset - [RCW15] and [NZdS⁺16].

6.3. Attention heatmaps

For better understanding of the results we visualized attention via heatmaps. X-axis corresponds to words in the original article and Y-axis corresponds to the generated summary. Heatmap pixel values represent attention weights a_{ij} of j -th source word and i -th target word on a scale of 0 to 1. Higher values are represented by brighter colors which allow visual recog-

inition of parts important for the model. For easier understanding, original text (labels on X-axis) is presented without lemmatization. Generated summary (labels on Y-axis) is presented as unchanged output from the model, with lemmatization. Because of variability in attention weights, the scale of heatmaps values is different, which allows for an easier visual pattern recognition.

Figure 6.1 shows attention heatmap at initial stage before training and as expected attention weights a_{ij} are initialized randomly, without any visible pattern.

Figure 6.1: Attention heatmap - initial, randomly initialized a_{ij} attention scores.

Second example of attention heatmap in Figure 6.2 shows summary result where model was focused on the first few words. Almost a whole summary consisted of only those words, throwing out all the other information. It is worth noticing the model started to lose some information like *Jordan* along summary creation. This is related to RNN issue with capability of handling long sequences [PABvM⁺14, CvMG⁺14]. Other words like *wodny* did not disappear in summary, because they reappeared several times in the rest of the source text and raised the value of attention weights.

Sometimes attention mechanism was focused mainly on a single word and still performed a decent summary. Figure 6.3 shows such a situation, when it was clearly biased towards a surname of a person. Considering this, the output summary was able to capture the most important facts, even transforming abbreviation *GP* into *grand prix*. This association is unfortunately not clearly visible in attention heatmap, because *GP* is even more related to the name and surname *Adrian miedziński* of a person, holding a *GP / granda prix* title. Such a result was possible due

Figure 6.2: Attention heatmap - losing information for long sequence

to the relatively high percentage of sports-related articles in the dataset, allowing the model to adapt to the topic better.

Potential of sequence-to-sequence mechanism in text summarization is well visible in Figure 6.4. Attention heatmap showed very high focus on name of a team *ksm* and resulted in adding supplementary information about team like its origin city *Krosno*. It was possible because of the information about the club obtained and preserved during the training process. On the other hand summary added extra information related to *Kraków*, a city mentioned in source text, but these were not related to the source text.

Next sample attention heatmap in Figure 6.5 shows a visible difference in attention weights magnitude between words related to *wypadek* (eng. *accident*) such as *ratownik* (eng. *paramedic*), *zginąć* (eng. *to die*), *tragedia* (eng. *tragedy*); and other words in the original text. Generated summary correctly focused on the article's main topic ignoring, however, the information present in further part. Model focused on specific facts presented at the beginning of a text, not vague details of a topic available later. As stated in work [OD11] this is a common issue for automated summaries in emergency or accident related articles.

An example of failed summary is shown in Figure 6.6. At the beginning, in the middle and at the end, model was uncertain about what it should pay the attention to. This is visible in numerous brighter columns on the heatmap. Model took the very first word from source text as

Figure 6.3: Attention heatmap - bias on named entity

Figure 6.4: Attention heatmap - extra information in summary

Figure 6.5: Attention heatmap - facts focused article

context and was unable to capture the sense of original text and connect it with any other word in its vocabulary. As a consequence, the model produced the list consisting of a single word.

Figure 6.6: Attention heatmap - failure with context

Chapter 7

Conclusion

In this dissertation we tackled the problem of abstractive text summarization for Polish language. Solution used neural networks based on sequence-to-sequence architecture with attention mechanism. Our experiments were held using potentially flawed dataset as described in 6.1. *Not a real summary* section. Even considering this, we obtained a decent results using ROUGE [Lin04] metric, comparing to the other trials [RCW15, NZdS⁺16] of automatic text summarization for the English language and with correct datasets, as shown in Table 6.3. To assess performance on Polish language dataset, we evaluated our model on The Polish Summaries Corpus [OK14] and compared with results presented in [Kop19], as available in Table 6.2. Comparison showed several times drop in metric scores caused by the usage of different vocabulary and bias obtained by the trained model from the custom dataset in training. This indicated the areas for improvement in the future in the view of dataset preparation and model training.

It is known that metrics for automatic text summarization are not very accurate in real life examples. There are many reasons for it, such as not taking into account synonyms or word order swaps. Some research even stated that the baseline solution, taking just the first words from the source text till the maximal specified length of a summary, outperformed many complex solutions [Kop15].

To overcome these issues, we evaluated a very small subsample of generated summaries manually using attention heatmaps, as shown in 6.3. *Attention heatmaps*. As expected, results were far from ideal, sometimes being hard to understand, including many repetitions [OD11]. However, this was observed mainly for longer sequences due to the poor capability of handling long dependencies. Apart from that, generated summaries were able to capture the most important, usually directly written information. Because categories of texts in the dataset were not equally distributed, summaries often consisted of words related to sport or daily news. Overall, the summaries were correct enough to get the main idea of a particular source text in most cases.

In order to improve the quality of summaries, it will be worth to focus on particular text

categories like scientific articles, sport, business, etc. It will improve results in two ways because different types of text require different types of details in the generated summary and the expected summaries will be more consistent for the whole dataset. To improve the quality of summaries for long texts, it will be worth using newer architectures like Transformer [VSP⁺17] that depends only on feed-forward networks and attention mechanism.

Bibliography

- [BCB15] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.
- [BG05] Emilia Branny and Marek Gajecski. Text summarizing in polish. *Computer Science*, 7, 01 2005.
- [BVJS15] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS’15, page 1171–1179, Cambridge, MA, USA, 2015. MIT Press.
- [CCD⁺14] Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Hrafn Loftsson, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors. *Proceedings of the Ninth International Conference on Language Resources and Evaluation, LREC 2014*, Reykjavík, Iceland, 2014. ELRA.
- [CG04] Marcin Ciura and Damian Grund. A System to Adapt Techniques of Text Summarizing to Polish. page 4, 2004.
- [CvMG⁺14] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics.
- [DBL08] Proceedings of the first text analysis conference, TAC 2008, gaithersburg, maryland, usa, november 17-19, 2008. NIST, 2008.

- [DSW08a] Adam Dudczak, Jerzy Stefanowski, and Dawid Weiss. Automatyczna selekcja zdań dla tekstów prasowych w języku polskim. Technical report, 2008.
- [DSW08b] Adam Dudczak, Jerzy Stefanowski, and Dawid Weiss. Comparing performance of text summarization methods on polish news articles. 2008.
- [Dud07] Adam Dudczak. Zastosowanie wybranych metod eksploracji danych do tworzenia streszczeń tekstów prasowych dla języka polskiego. 2007.
- [Elm90] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.
- [HKG⁺15] Karl Moritz Hermann, Tomáš Kočiský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS’15, page 1693–1701, Cambridge, MA, USA, 2015. MIT Press.
- [Hoc98] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6:107–116, 04 1998.
- [HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.
- [HSK⁺12] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *ArXiv*, abs/1207.0580, 2012.
- [JP15] Krzysztof Jassem and Łukasz Pawluczuk. Automatic summarization of Polish news articles by sentence selection. In *2015 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 337–341, September 2015.
- [KB13] Nal Kalchbrenner and Phil Blunsom. Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1700–1709, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.
- [KB14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

- [Kir07] David Kirk. Nvidia cuda software and gpu parallel computing architecture. In *Proceedings of the 6th International Symposium on Memory Management, ISMM '07*, page 103–104, New York, NY, USA, 2007. Association for Computing Machinery.
- [KL16] Dimitrios Kouzis-Loukas. *Learning Scrappy*. Packt Publishing Ltd, 2016.
- [Kop15] Mateusz Kopeć. Coreference-based content selection for automatic summarization of polish news. *Selected problems in information technologies*, pages 23–46, 2015.
- [Kop19] Mateusz Kopeć. Three-step coreference-based summarizer for Polish news texts. *Poznan Studies in Contemporary Linguistics*, 55(2):397–443, June 2019. Publisher: De Gruyter Mouton Section: Poznan Studies in Contemporary Linguistics.
- [Lin04] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics.
- [LSMS18] Junyang Lin, Xu Sun, Shuming Ma, and Qi Su. Global encoding for abstractive summarization. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 163–169, Melbourne, Australia, July 2018. Association for Computational Linguistics.
- [Luh58] H. P. Luhn. The automatic creation of literature abstracts. *IBM J. Res. Dev.*, 2(2):159–165, April 1958.
- [NCL18] Shashi Narayan, Shay B. Cohen, and Mirella Lapata. Don’t give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1797–1807, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.
- [NZdS⁺16] Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Çağlar Gülçehre, and Bing Xiang. Abstractive text summarization using sequence-to-sequence RNNs and beyond. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 280–290, Berlin, Germany, August 2016. Association for Computational Linguistics.

- [OD11] Karolina Owczarzak and Hoa Trang Dang. Who wrote what where: Analyzing the content of human and automatic summaries. In *Proceedings of the Workshop on Automatic Summarization for Different Genres, Media, and Languages*, WASDGM '11, page 25–32, USA, 2011. Association for Computational Linguistics.
- [ODH07] Paul Over, Hoa Dang, and Donna Harman. Duc in context. *Inf. Process. Manage.*, 43(6):1506–1520, November 2007.
- [OK14] Maciej Ogrodniczuk and Mateusz Kopec. The Polish Summaries Corpus. page 4, 2014.
- [PABvM⁺14] Jean Pouget-Abadie, Dzmitry Bahdanau, Bart van Merriënboer, Kyunghyun Cho, and Yoshua Bengio. Overcoming the curse of sentence length for neural machine translation using automatic segmentation. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 78–85, Doha, Qatar, October 2014. Association for Computational Linguistics.
- [PGM⁺19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019.
- [PPR19] *Proceedings of PP-RAI 2019 Conference*, Wrocław, 2019. Department of Systems and Computer Networks, Faculty of Electronics, Wrocław University of Science and Technology.
- [PRWZ02] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: A method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, page 311–318, USA, 2002. Association for Computational Linguistics.
- [RCW15] Alexander M. Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 379–389, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

- [RHW88] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. *Learning Representations by Back-Propagating Errors*. MIT Press, Cambridge, MA, USA, 1988.
- [RWJ⁺94] Stephen Robertson, Steve Walker, Susan Jones, Micheline Hancock-Beaulieu, and Mike Gatford. Okapi at trec-3. pages 0–, 01 1994.
- [SLM17] Abigail See, Peter J. Liu, and Christopher D. Manning. Get to the point: Summarization with pointer-generator networks. *ArXiv*, abs/1704.04368, 2017.
- [SM86] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., USA, 1986.
- [SP13] Horacio Saggion and Thierry Poibeau. *Automatic Text Summarization: Past, Present and Future*. 01 2013.
- [SVL14] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014.
- [TK19] Ryszard Tuora and Łukasz Kobylński. Integrating Polish language tools and resources in Spacy, oct 2019.
- [VFJ15] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS’15, page 2692–2700, Cambridge, MA, USA, 2015. MIT Press.
- [VSP⁺17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *ArXiv*, abs/1706.03762, 2017.
- [Wei01] Dawid Weiss. Korpus rzeczpospolitej. [on-line] <http://www.cs.put.poznan.pl/dweiss/rzeczpospolita>, 2001.
- [Wol14] Marcin Woliński. Morfeusz reloaded. In Calzolari et al. [CCD⁺14], pages 1106–1111.
- [ZHSJ20] Jingzhao Zhang, Tianxing He, Suvrit Sra, and Ali Jadbabaie. Why gradient clipping accelerates training: A theoretical justification for adaptivity. In *International Conference on Learning Representations*, 2020.
- [ZLG95] Rolf A. Zwaan, Mark C. Langston, and Arthur C. Graesser. The construction of situation models in narrative comprehension: An event-indexing model. *Psychological Science*, 6(5):292–297, 1995.

- [ZLLG16] Renxian Zhang, Wenjie Li, Naishi Liu, and Dehong Gao. Coherent narrative summarization with a cognitive model. *Computer Speech & Language*, 35:134 – 160, 2016.
- [Św10] Joanna Świetlicka. Metody maszynowego uczenia w automatycznym streszczeniu tekstów [machine learning methods in automatic text summarization], master's thesis. 2010.

List of Figures

4.1	Model architecture overview	24
4.2	Recurrent neural network (RNN)	25
4.3	Gated Recurrent Unit (GRU)	26
4.4	Bidirectional recurrent neural network	28
4.5	Dropout layer visualization	28
4.6	Bahdanau attention visualization	31
6.1	Attention heatmap - initial, randomly initialized a_{ij} attention scores.	39
6.2	Attention heatmap - losing information for long sequence	40
6.3	Attention heatmap - bias on named entity	41
6.4	Attention heatmap - extra information in summary	41
6.5	Attention heatmap - facts focused article	42
6.6	Attention heatmap - failure with context	43

List of Tables

- 3.1 Datasets comparison 17
- 3.2 Dataset sizes 18
- 3.3 Dataset cleaning rules 19
- 3.4 Numerical types replacement 19

- 4.1 Model learnable parameters comparison. 23

- 6.1 ROUGE-1, ROUGE-2, ROUGE-L scores comparison on test set. 38
- 6.2 Comparison with extraction-based methods on Polish Summaries Corpus. . . . 38
- 6.3 ROUGE metrics F_1 scores trials comparison. 38

Appendix A

Hadamard product

Hadamard product is element-wise multiplication. Its operation is illustrated for matrices A and B in equation (A.1).

$$\begin{aligned} A &= \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \\ B &= \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix} \\ A \odot B &= \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \odot \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix} = \begin{bmatrix} a_{11} * b_{11} & a_{12} * b_{12} \\ a_{21} * b_{21} & a_{22} * b_{22} \\ a_{31} * b_{31} & a_{32} * b_{32} \end{bmatrix} \end{aligned} \quad (\text{A.1})$$

Source code availability

Source code is available in Github and Gitlab repositories under same version:

https://github.com/oziomekl/NLP_summarizer/tree/master/NLPer

https://gitlab.com/oziomekl/nlp_sandbox/-/tree/master/NLPer

Project documentation available at:

<https://nlper.readthedocs.io/>