EXTRACTIVE SUMMARIZATION USING SENTENCE EMBEDDINGS

Automatic summarization of news articles at Blendle

Lucas de Haas
3667898

Master of Artificial Intelligence
Department of Information and Computing Sciences
Faculty of Science
Utrecht University

November 29, 2017

Utrecht University

Lucas de Haas: *Extractive summarization using sentence embeddings,* Automatic summarization of news articles at Blendle, November 29, 2017

SUPERVISORS:
Dr. Daan Odijk (Blendle)
Drs. Martijn Spitters (Blendle)
Dr. Ad Feelders (Utrecht University)
Prof. Dr. Arno Siebes (Utrecht University, second reader)

ABSTRACT

In this thesis, we investigate techniques for automatically extracting the most informative subset of sentences, also known as *extractive summarization*. We specifically focus on techniques using word embeddings to construct semantically aware sentence representations.

The thesis is comprised of three studies. In the first study, we perform an experimental survey, comparing embedding-based extractive summarization methods with commonly used baselines. We find that the embedding-based methods do not necessarily outperform common baselines. We also find that the strong beginning-of-article heuristic (called *LEAD*) outperforms all other summarization methods. In the second study, we substitute various sentence embeddings into the non-embedding-based TextRank method, in order to evaluate the added value of embeddings in this case. Although the new method does not outperform the original TextRank method, it does perform on par with it using substantially less preprocessing. In the third study, we propose a summarization method based on a recurrent neural network (RNN) architecture, which is the only summarization method that outperforms the *LEAD* baseline. This model is an adaptation of the model by Cheng and Lapata (2016). In order to make the RNN training more flexible, we further propose a semi-supervised training framework for this RNN architecture by using unsupervised methods for pre- or co-training the RNN summarizer.

The main contributions of this thesis are as follows. Firstly, our experimental survey compares previously proposed embedding-based methods with common baselines. Such a comparison has not been performed before. Secondly, we train both a Dutch and English word2vec model on Blendle's news article database. For English, this model performs on par with the Google News vectors, even though our model is much smaller. For Dutch, the model might even be best-performing Dutch word2vec model due Blendle's uniquely large and clean training set. Finally, our proposed RNN-based semi-supervised framework is capable of learning and combining the behavior of various extractive summarization methods.

CONTENTS

1 INTRODUCTION 1

2 GENERAL BACKGROUND 5

2.1 Preprocessing . . . 6

2.1.1 Cleaning . . . 6

2.1.2 Sentence boundary detection . . . 6

2.1.3 Tokenization . . . 7

2.1.4 Stemming/lemmatization . . . 7

2.1.5 Stop word & POS tag filtering . . . 7

2.2 Text representation . . . 8

2.2.1 Sentence similarities . . . 8

2.2.2 Building blocks: (sub)word embeddings . . . 9

2.2.3 Sentence embeddings . . . 13

2.3 Sentence selection . . . 16

2.3.1 Maximal marginal relevance . . . 16

2.3.2 Centroid-based: MEAD . . . 17

2.3.3 Graph-based: TextRank & LexRank . . . 17

2.3.4 Greedy submodular optimization . . . 18

2.3.5 Supervised sentence classifiers: recurrent neural networks . . . 19

2.4 Evaluation: Rouge . . . 19

2.4.1 ROUGE shortcomings . . . 21

3 STUDY 1: A SURVEY OF EXISTING EMBEDDING-BASED SUMMARIZATION METHODS 22

3.1 Embedding-based summarization methods & common baselines 23

3.1.1 The beginning-of-article baseline: LEAD . . . 23

3.1.2 Lin & Bilmes (2010, 2011): greedy submodular baselines 23

3.1.3 Kågebäck et al. (2014): enriching Lin & Bilmes (2011) 25

3.1.4 Kobayashi et al. (2015): proposing alternative objective functions . . . 25

3.1.5 Yogatama et al. (2015): maximizing semantic volume 26

3.2 Datasets & word2vec models . . . 27

3.2.1 DUC-2002 . . . 27

3.2.2 TAC-2008 . . . 28

3.2.3 Opinosis . . . 28
3.2.4 Word2vec models . . . 28
3.3 Experiment . . . 29
3.4 Results & discussion . . . 31
4 STUDY 2: ENRICHING TEXTRANK WITH WORD EMBEDDINGS 34
4.1 The TextRank algorithm . . . 34
4.2 Reproduction of TextRank results . . . 36
4.3 Experiment . . . 36
4.4 Results & discussion . . . 38
5 STUDY 3: A NEW ALTERNATIVE: RECURRENT NEURAL NETWORKS 41
5.1 The challenge: outperforming LEAD . . . 41
5.2 Recurrent neural networks for summarization . . . 42
5.2.1 RNN architectures: background . . . 42
5.2.2 Our architecture . . . 48
5.3 Dataset . . . 49
5.4 Experiment & results . . . 50
5.4.1 General settings (hyperparameters, layer sizes, etc.) . . . 50
5.4.2 Run 1: Normal training . . . 52
5.4.3 Run 2–6: Co-/pre-training with TextRank labels . . . 55
5.5 Discussion . . . 58
6 DISCUSSION AND CONCLUSION 59
6.1 Discussion of research questions . . . 60
6.2 Future directions . . . 61
A FEASIBILITY OF RNN-BASED SUMMARIZATION ON DUTCH BLENDLE ARTICLES 67
A.1 Dataset construction . . . 67
A.2 Word2vec model . . . 68

LIST OF FIGURES

Figure 1 Example of an article presentation from Blendle's platform. 1
Figure 2 Example of what a highlighted article could look like. 3
Figure 3 Schematic overview of the elements of an extractive summarization method. 5
Figure 4 Schematic representation of both the CBOW and skip-gram word2vec model. Although the context words seem to be ordered in this figure, position is disregarded during training; the context is treated as a bag of words. Image adopted from the original paper (Mikolov et al., 2013a). 10
Figure 5 Example of the target and context words for a sentence, with a window size of 2. The window around the target word defines the set of context words used for training the skip-gram model. Adopted from McCormick (2016). 11
Figure 6 The conceptual idea of maximizing the semantic volume, simplified to a two-dimensional space. Sentences are represented as dots in the 2-D vector space. Given some length restriction, the optimal solution for these 7 sentences is the set indicated in red, spanning the largest semantic volume. Adopted from Yogatama et al. (2015). 27
Figure 7 Static visual representation of the PageRank algorithm. The more arrows are pointing to a node, the more important (i.e. bigger) it becomes. The color of an arrow indicates its weight, which corresponds to the node size. Note that in the TextRank case, all relations are symmetric. 35
Figure 8 Normalized positional distribution of selected sentences per article. The number indicates the absolute number of sentences falling in that bin. 42

Figure 9 An 'unrolled' representation of a recurrent neural network. Note that all cells share the same weights; only the input per time step is different, and partly dependent on the previous step. Adopted from Olah (2015). 43
Figure 10 The internal structure of an RNN. Adopted from Olah (2015). 43
Figure 11 The internal structure of an LSTM network. Adopted from Olah (2015). 44
Figure 12 The encoder-decoder architecture. Adapted image from lecture. 45
Figure 13 The extractive summarization architecture proposed by Cheng and Lapata (2016). Image is an adapted version from the original paper. 47
Figure 14 Normalized positional distribution of 1-labels in a random sample of 10000 articles from the DailyMail dataset. The number indicates the the probability of a sentence to occur in that position (of 20 position bins). 50
Figure 15 Plot of the relation between mean and loss (left) and the loss over training iterations (right) for the first, normal training run with the Blendle word2vec model. Run on the Google model is very comparable. Data points are means over every 1000 iterations. . . 52
Figure 16 Normalized positional distribution of selected sentences per article for the normally trained RNN models, with the *LEAD* baseline and TextRank for reference. The number indicates the absolute number of sentences falling in that bin. 53
Figure 17 Plot of the relation between mean and loss (left) and the loss over training iterations (right) on the TextRank pre-training with the Blendle model. Google pre-training is very comparable. Data points are means over every 1000 iterations. 55

Figure 18 Normalized positional distribution of selected sentences per article for the all RNN models (both Google and Blendle), with the *LEAD* baseline and TextRank for reference. The number indicates the absolute number of sentences falling in that bin. Marked distributions achieve the best performance on DUC-2002 (see Table 10). 57

LIST OF TABLES

Table 1 The means and standard deviations ROUGE F-scores between reference summaries on the DUC-2002 dataset, for all articles with two reference summaries. 21

Table 2 Rouge scores on the Opinosis, TAC-2008, and DUC-2002 datasets for various algorithms. Results for both Google's word2vec model and the word2vec model trained on Blendle's data are reported. Note that the scores for TAC-2008 and DUC-2002 are Rouge F-measures, while the scores on Opinosis are Rouge R-measures. For DUC-2002, the best scores behind *LEAD* are italicized. 32

Table 3 The performance of default TextRank in different preprocessing configurations on the DUC-2002 dataset. 36

Table 4 The performance of default TextRank in different preprocessing configurations compared with word2vec-enriched TextRank on the DUC-2002 dataset, using Google's and Blendle's news vectors. 37

Table 5 System summaries for the raw version of TextRank, the complete version, and the IDF-reweighted word2vec-based TextRank method on an article from the DUC-2002 dataset. Sentence positions are reported for every system summary. Reference summaries are also shown for comparison. Note that both word2vec models result in the same summary in this case. The *LEAD* baseline from Chapter 3 is also shown. For this article, the embedding-based version of TextRank detects the importance of sentence 1, whereas the original TextRank does not. 40
Table 6 ROUGE F-scores of best-performing TextRank methods and the *LEAD* baseline on the DUC-2002 dataset. *LEAD* significantly outperforms all other methods. . 41
Table 7 An overview of layer sizes in the RNN model. 51
Table 8 ROUGE F-scores of the first RNN-based model and the strongest SDS baselines on the DUC-2002 dataset. 52
Table 9 System summaries for the complete version of TextRank, the first RNN ranker (both the Google en Blendle word2vec version), and the *LEAD* baseline on an article from the DUC-2002 dataset. Reference summaries are also shown for comparison. The RNN rankers seem to learn which sentences to skip when reading the article. 54
Table 10 ROUGE F-scores on all different RNN model versions and the *LEAD* baseline. 56
Table 11 Some ramdomly sampled article intros. 68

1 INTRODUCTION

At Blendle[1], many articles from various news sources come in every day. An editorial team is responsible for selecting articles from this daily news stream that are worth reading. This set of picked articles is then ranked for every user separately, in order to select a 'bundle' from the set that hopefully matches the user's interest. This personalized bundle of articles is then sent to the user in a daily newsletter and shown on the personal timeline in the Blendle app. See Figure 1 for an example of the presentation of an article on Blendle's platform.

In order to present an article as done in Figure 1, a representative 'caption' is needed. Until recently, all captions were written by the editorial team. A problem with the current approach, however, follows from the size and diversity of the set of 'picked' articles by the editorial team. Due to time constraints, the editors are bound to pick certain types of articles that will probably be read by a large number of users. Although this covers a lot of the users, a bigger, more diverse set of articles for the personalization algorithm to select from would enhance the user's experience. This is why *autopicks* (i.e. automatically picked articles) were recently introduced. Complementary to the articles picked by the editors, a broader set of possibly interesting

Figure 1: Example of an article presentation from Blendle's platform.

1 https://blendle.com/

articles is automatically selected. This set is then curated by the editors to make sure that all automatically picked articles are of high quality.

As with the editorial picks, these autopicks must be presented to Blendle's users by accompanying every article with a caption containing the article's core information. Generating a caption containing the core information of an article is very similar to the well-known problem of automatic summarization.

Abstractive summarization (i.e. generating newly formed sentences, capturing the essence of a text) techniques are not (yet) capable of reliably forming grammatical sentences; they are therefore not in the scope of the current research. Another research direction in automatic summarization aims to select the most informative subset of sentences from a text. This is known as *extractive summarization*. By selecting sentences from the original text, the correctness of the sentences themselves is ensured. Using extractive summarization, the core information of any possibly interesting article could be automatically extracted and used directly on the platform.

Besides the aforementioned use case of generating captions for autopicks, extractive summarization of articles provides another opportunity to improve Blendle's editorial workflow. The same summarization algorithm could automatically highlight the most important sentences in order to improve the reading experience and reading speed for the editorial team on any article. An example of what a highlighted article could look like is shown in Figure 2.

In brief, the extraction of core information from articles — now done by hand for a small subset of the articles — should be automated, in order to be able to personalize for Blendle's users from a much bigger set of articles. Extractive summarization could help by generating or suggesting captions for articles, and by automatically highlighting articles to improve the editorial workflow. In this thesis, we will specifically focus on extractive summarization techniques using sentence embeddings as their basis. As explained in Chapter 2, sentence embeddings are unsupervisedly trained on large text corpora to continuously represent the semantic information, thereby adding semantic information for free. Since their popularization in 2013 (Mikolov et al., 2013a), the unsupervised information added by embeddings changed the whole NLP research field.

The objective for this thesis is to *design, implement, and evaluate an embedding-based algorithm for extractive summarization, in order to auto-*

Robbe Geysmans. (rr)

(iStockphoto)

Fairtrade doet het goed. De omzet van fairtrade-producten steeg vorig jaar in België met bijna een kwart en je vindt de producten tegenwoordig overal: zeker fairtradebananen en -koffie liggen in bijna elke supermarkt.

Als Fairtrade het goed doet, dan de Wereldwinkels ook, zou je denken. Tenslotte waren het de Oxfam-Wereldwinkels die eerlijke handel op de kaart zetten in België. Dus als mensen hun ethisch kompas vaker aanspreken tijdens het winkelen, dan moet de Wereldwinkel daar toch de vruchten van plukken? **Toch niet, want verschillende Wereldwinkels zien hun verkoopcijfers dalen.**

Robbe Geysmans en Lesley Hustinx (UGent) verbaasden zich over die tegenstelling en zochten uit hoe winkelende mensen de keuze voor Fairtrade maken.

Hoe ethisch is de fairtrade- consument?

Robbe Geysmans: 'Niet zo ethisch als we denken. Je gaat ervan uit dat hij weet waarvoor Fairtrade staat, en dat hij om die reden bewust voor een fairtradeproduct kiest. In de praktijk valt dat tegen. **Toen ik in Wereldwinkels mensen bevroeg, was een op de drie onzeker over de betekenis van Fairtrade. De informatiebrochures werden ook amper bekeken.** Consumenten leggen de verantwoordelijkheid liever bij de organisatie: de Wereldwinkel weet wat goed is, dus alle producten in de winkel zijn goed en ik hoef me daarover

Figure 2: Example of what a highlighted article could look like.

matically find the most important set of sentences in any article. To fulfill this objective, this thesis addresses three research questions:

1. How do the current embedding-based extractive methods compare to each other, and to commonly used non-embedding-based baselines?

2. Is it possible to improve the performance of existing non-embedding-based methods by substituting embeddings?

3. Can a sequential model using unsupervised word embeddings improve on the strong beginning-of-article heuristic?

The rest of this thesis is structured as follows. In Chapter 2, we discuss relevant background of all components involved in extractive summarization. In Chapter 3, we will focus on Research Question 1 by performing an experimental survey comparing existing embedding-based methods and commonly

used baselines. In Chapter 4, we discuss Research Question 2 by attempting to increase the performance of a strong baseline method, TextRank, by substituting various sentence embeddings. In Chapter 5, we attempt to answer Research Question 3, training recurrent neural networks for extractive summarization. In Chapter 6, we discuss all research questions and results and give a conclusion, also providing suggestions for further research.

The main contributions of this work are as follows. Firstly, we provide a thorough comparison of previously proposed, embedding-based extractive summarization methods against commonly used baselines. To our knowledge, such a comparison has not been performed before. Secondly, we train both a Dutch and English word2vec model on Blendle's news article database. For English, this model performs on par with the Google News vectors, even though our model is much smaller. For Dutch, our model might even be best-performing Dutch word2vec model due to Blendle's uniquely large and clean training set.[2] Finally, we propose a RNN-based semi-supervised framework capable of learning and combining the behavior of various extractive summarization methods.

2 The Dutch Blendle word2vec model is shortly discussed in Appendix A.

2

GENERAL BACKGROUND

Extractive summarization techniques attempt to select the most informative subset of sentences from a text given a length constraint, as mentioned in Chapter 1. Extractive summarization tasks are either aimed at summarizing single documents or clusters of related documents. These tasks are respectively named single document summarization (SDS) and multi-document summarization (MDS). Every extractive summarization algorithm consists of 4 basic components, which are schematically represented as a summarization 'pipeline' in Figure 3. This pipeline consists of a preprocessing, representation, and selection phase. Although evaluation is not a strictly necessary phase for summarization, it could be considered part of the pipeline.

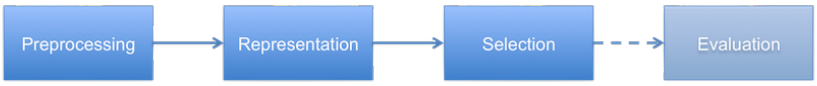

Figure 3: Schematic overview of the elements of an extractive summarization method.

Summarization systems are usually also quite modular: in most cases, it is possible to inject another sentence representation into a proposed summarization method. In the sections that follow, a general overview of all four summarization steps is given. For the sentence representation step, this overview goes far beyond the representation techniques used in the current field of extractive summarization, in order to give an accurate overview of text representation in the entire NLP field. Note that for every step in the summarization pipeline, the aim is to give an overview of that step exclusively; no full summarization pipelines are discussed in this chapter. In this background, we do not discriminate between MDS and SDS methods, although formally, only the latter is relevant to the goal of this thesis. Some important complete summarization methods used later in this thesis are explained in Chapters 3 and 4.

2.1 PREPROCESSING

As with many natural language processing problems, preprocessing is necessary to facilitate sentence representation. Necessary preprocessing steps are sentence boundary detection and tokenization in order to respectively find sentence and word boundaries in the text. In Blendle's preprocessing pipeline, all text is already cleaned before these necessary steps. In many extractive summarization methods, additional preprocessing is performed. Commonly used preprocessing steps include stemming/lemmatization, stop word filtering, and part-of-speech (POS) tag filtering. We will describe all of these preprocessing steps below.

2.1.1 *Cleaning*

In Blendle's article enrichment pipeline, which is used to extract features from raw article text, articles are cleaned before further preprocessing is applied. The cleaner removes Markdown and HTML tags if these are in the content, and transforms rare punctuation (especially all different quotes) into its default version.

2.1.2 *Sentence boundary detection*

Sentence boundary detection, also known as sentence splitting or sentence tokenization, is the problem of deciding where sentences end. There is a straightforward solution in many cases (i.e. in most cases, splitting after a period or question/exclamation mark is the right decision). Ironically, however, this is not true for the last sentence: abbreviations, quotations including punctuation, and many other situations make this task far from trivial. Although sentence splitters can be trained unsupervisedly to learn exceptions, no model is flawless. In Blendle's pipeline, a standard, pre-trained[1] sentence splitter from the NLTK toolkit is used. For Dutch, the splitter has been improved by appending a scraped list of abbreviations to the exception list. This sentence splitter is used as preprocessing for all reproductions of summarization methods described later in this thesis.

1 Separately pre-trained for German, English, and Dutch

2.1.3 *Tokenization*

Word tokenization is the problem of splitting a sentence into separate word tokens. While splitting on whitespace is an excellent heuristic, this approach fails in many cases. Quotation marks should be interpreted as separate tokens, for example, even though they are not whitespace-separated from the words they mark. In our pipeline, we use the Pattern tokenizer[2] developed by the computational linguistics institute CLiPS in Antwerpen, which has pre-trained models for all three languages currently on Blendle's platform (Dutch, English, and German). Although this tokenizer has been designed to find both sentence and word boundaries, we only use the latter functionality.

2.1.4 *Stemming/lemmatization*

Stemming and lemmatization are different approaches to the problem of reducing all different inflectional forms of a word to a common 'base form'. Stemming is a fast, but crude approach: it heuristically cuts off all affixes, not considering the word's context, or a vocabulary of known base forms. More often than not, stems are incomplete words. Lemmatization, on the other hand, is computationally more demanding, but generally achieves higher precision. Typically based on a large database, lemmatization is the process of looking up the 'lemma' of a word (or word-POS tag combination). Using the commonly used Porter stemmer, for example, the word 'are' would become 'ar', whereas using a lemmatizer, the result would become 'be'.

Blendle's pipeline contains the classic, heuristic Porter stemmer (with separate heuristics per language) provided in the NLTK toolkit. As this stemmer is usually adopted in papers on extractive summarization, and stemming is only used for reproducing earlier papers, this is the only stemming/lemmatization method that is considered throughout this thesis.

2.1.5 *Stop word & POS tag filtering*

Both stop word and POS tag filtering are meant to filter out certain words that might not be relevant to the task at hand. Stop word filtering filters the most common, and thereby least informative words. This often includes

2 See `https://www.clips.uantwerpen.be/pages/pattern-en`

articles, inflections of 'be' and other common verbs, personal pronouns, etc. Blendle's stop word list contains 625 strings in total.

POS tag filtering is the more systematical filtering process of removing all words that have a certain syntactical function. POS tagging is the NLP task of labelling each word in a sentence with its type. Common types include (singular and plural) nouns, verbs, adverbs etc. By filtering out all POS tags except a few that might capture certain information useful for the task at hand, textual data can be sanitized. This is a crude approach, however; a lot of possibly important words could be excluded from further analysis.

Generally, this could be said about applying more than the necessary preprocessing (i.e. sentence splitting & tokenization): if more preprocessing is performed, more potentially important information could be removed from the input.[3] This is why we prefer to generally avoid additional preprocessing, apart from sentence splitting and word tokenization. We would rather train a model that is expressive enough to learn ignoring unimportant data, than to remove it ourselves. This follows the current trend in NLP research, and even machine learning research in general: already processing raw data by preprocessing or even feature engineering, instead of directly training a model usually leads to worse results.

2.2 TEXT REPRESENTATION

The initial aim of text representation for extractive summarization was to construct a similarity measure between sentences in the document. Many sentence selection methods (especially graph-based methods; see Section 2.3) rely on similarities between all sentences in a text.

2.2.1 *Sentence similarities*

Initially, simple word or phrase overlap between sentences were proposed as similarity measures (Banerjee and Pedersen, 2003; Metzler et al., 2005; Mihalcea and Tarau, 2004). Although these measures may correctly score sentence similarity in some cases, they do not account for the fact that some words are less informative than others. Sentence similarity measures based

3 Although POS tagging only adds information, POS tag filtering removes words from the data.

on term frequency-inverse document frequency (*TF-IDF*) vectors solved this issue to a certain extent (Erkan and Radev, 2004). TF-IDF is a simple, but powerful representation of sentences, indicating the relative importance of words in a sentence. Cosine similarity between TF-IDF vectors is still a popular baseline measure for its simplicity, efficiency, and effectiveness.

Currently, the enormous number of sentence similarity measures could generally be divided into two groups. The first group comprises measures which are designed only with the task of similarity measurement in mind. At SemEval[4], a yearly event for evaluation of computational semantic analysis systems, top-performing measures are built and tuned specifically for the task of sentence similarity (Sultan et al., 2015), reaching the current state-of-the-art performance on this specific yearly dataset. These methods (e.g. the winning method of SemEval 2016 (Brychcın and Svoboda, 2016)) are usually based on simple word alignment, in this case combined with TF-IDF weights to control for word importance. Although achieving state-of-the-art performance on the sentence similarity task, these methods are merely aimed at this specific task, hence not providing a more generalized framework for sentence semantics. The similarity measure does not necessarily lead to a robust representation of a sentence. Besides, these tasks (e.g. the SemEval sentence similarity task) tend to oversimplify reality as the datasets usually consist of relatively short, syntactically simple sentences as compared to real news corpora. Similarity between simpler sentences might be easier to evaluate than similarity between longer, more complex sentences, hence the SemEval task does not necessarily evaluate sentence representation in a real, in our case news-related setting.

The second group of methods does not solely aim at the task of measuring similarity, but on a more robust semantic representation of sentences per se. These semantic representations are usually based on word-level semantic representations. This group of more robust representations is discussed below.

2.2.2 *Building blocks: (sub)word embeddings*

As sentences are constructed with words, this is an essential building block for sentence embeddings, used for all sentence embedding papers discussed in the next paragraph. The idea that similar words occur in similar con-

4 e.g. `http://alt.qcri.org/semeval2014/task1/`

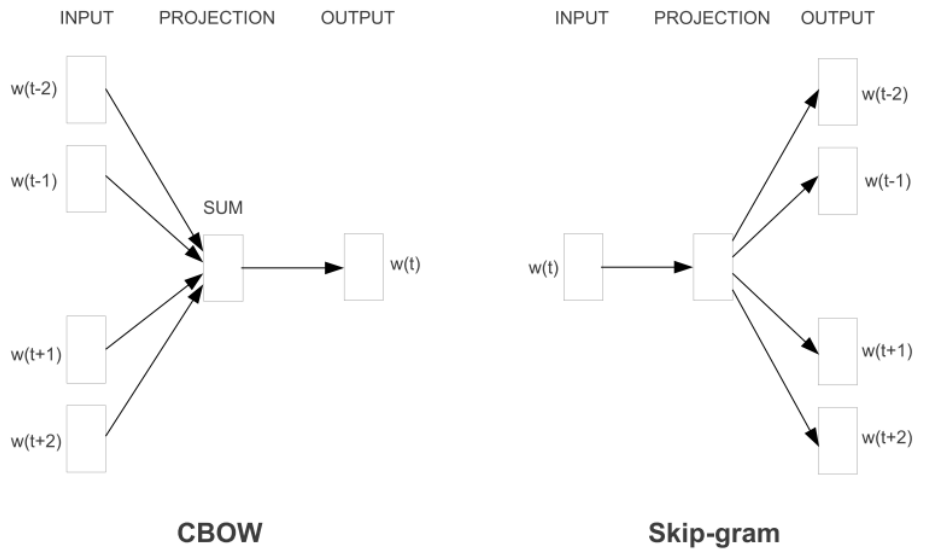

Figure 4: Schematic representation of both the CBOW and skip-gram word2vec model. Although the context words seem to be ordered in this figure, position is disregarded during training; the context is treated as a bag of words. Image adopted from the original paper (Mikolov et al., 2013a).

texts, coined the Distributional Hypothesis (Harris, 1954) has become the dominant approach in semantic representation: distributed representations of words have become the foundation of many modern solutions for NLP problems. The simple idea of representing a word as a prediction of its context, thereby yielding a continuous representation of a word embedded in a vector space, turns out to be a very robust representation of a word's meaning. Various unsupervised methods towards continuous word embeddings are available. The most popular and well-explored method is *word2vec* (Mikolov et al., 2013a). It naturally captures linguistic relations such as (dis)similarity and word analogies, both in the semantic and syntactic sense (Mikolov et al., 2013b). As word2vec is the word representation method used throughout this thesis, we will now discuss it in more detail.

Word2vec (Mikolov et al., 2013a) is developed by Google, and comprises two different approaches for training word embeddings: the continuous bag-of-words (*CBOW*) approach and the *skip-gram* approach. Both are schematically represented in Figure 4. In short, the task of the CBOW model is to

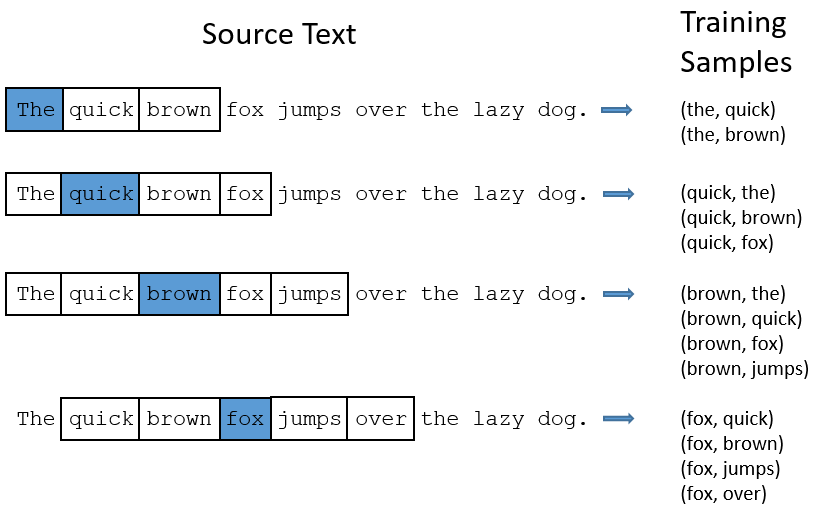

Figure 5: Example of the target and context words for a sentence, with a window size of 2. The window around the target word defines the set of context words used for training the skip-gram model. Adopted from McCormick (2016).

predict a word given its context, whereas for skip-gram, the task is to predict which words are in the context given a target word. The context is defined as the 'window' of words around a target word, where typical window sizes are between 2 and 5 words. This is illustrated in Figure 5. Note that we do not consider the word order of the context; as the context is a bag of words, the task is merely to predict whether a word is in the context, disregarding its position. As we will only use the skip-gram model throughout this thesis, we will solely discuss this model from now on.

Note that word2vec is always trained on a large corpus of tokenized sentences. For skip-gram, the general steps are as follows. First, we build a vocabulary by passing over the corpus once, keeping all words that occur more than n times. Then we initialize two random vectors of arbitrary dimensionality (typically 300) for each word in the vocabulary: a 'target' vector representing its meaning, and a 'context' vector to represent the separate 'context meaning' of a word. Combining all vectors in a matrix, we thus have two $|\mathcal{V}| \times D$ matrices, where D is the number of dimensions, and $|\mathcal{V}|$ is the number of words in the vocabulary. We name these matrices the *embedding matrix*, containing the actual embeddings of each word, and the *context matrix*, containing the 'context embeddings' of each word. The goal is to train the initially random D-dimensional target vectors to predict their

context, thereby representing the target word's meaning. For every training iteration, we now sample a target word w_t from the corpus, estimate the probabilities of the context words w_c given the target word w_t:

$$p(w_c|w_t) = \frac{\exp\left(u_{w_c}^\top v_{w_t}\right)}{\sum_{w \in \mathcal{V}} \exp\left(u_w^\top v_{w_t}\right)},$$

where $\mathcal{V}$ is the set of all words in the vocabulary, u vectors are from the context matrix, and v vectors are from the embedding matrix. Hence the estimation of the context word probability is simply the inner product of a target vector and a context vector. Note that estimating these context word probabilities requires computing the full probability distribution over the vocabulary. A function that normalizes a set of predicted scores into a probability distribution, such as the function above, is typically called the *softmax* function. After computing the context probabilities, we backpropagate the prediction error through the target and context vectors using gradient descent, in order to maximize the log probability over the set of context words $\mathcal{C}_{w_t}$

$$\sum_{w_c \in \mathcal{C}_{w_t}} \log p(w_c|w_t).$$

In every iteration, we backpropagate through $|\mathcal{C}_{w_t}|$ words in the context matrix and one word in the embedding matrix, such that the target word predicts its context words. The complete objective of the skip-gram model is to maximize the average log-probability over the set of all target words $\mathcal{T}$

$$f_{obj}(u, v) = \frac{1}{|\mathcal{T}|} \sum_{w_t \in \mathcal{T}} \sum_{w_c \in \mathcal{C}_{w_t}} \log p(w_c|w_t).$$

As the corpus usually contains hundreds of thousands of words, this approach is very inefficient in practice, due to the fact that all word probabilities are estimated in every iteration. To overcome this issue, Mikolov et al. (2013b) introduced a loss function based on 'negative sampling', instead of computing the full distribution. Instead of computing the softmax over the whole vocabulary, we sample a few words that do *not* occur in the current context (usually $\sim$5), and modify our maximization objective to be a binary classification task that, using a sigmoid function, estimates whether a word belongs

in the context or not. For every iteration, our more efficient maximization objective now looks as follows:

$$f_{obj}(u,v) = \sum_{w_c \in \mathcal{C}} \left[\log \sigma(u_{w_c}^\top v_{w_t}) + \sum_{w_n \in \mathcal{N}} \log \sigma(-u_{w_n}^\top v_{w_t}) \right],$$

where $\mathcal{N}$ is the set of negative samples. By backpropagating the prediction error on the objective through the target, context and negative sample vectors, we update the model in a similar way, while being much more efficient. In the negative sampling case, we adjust one word in the embedding matrix and $|\mathcal{C}| + |\mathcal{N}|$ words in the context matrix.

After many iterations, the embedding matrix consists of meaningful vector representations for every word in the model vocabulary. This embedding matrix is the final result of the model.

Other popular (but comparable) word embedding methods are GloVe and FastText. *GloVe* (Pennington et al., 2014) is a count-based (as opposed to prediction-based) method that counts word occurrences in a word's context. This naturally leads to a co-occurrence matrix, whose dimensionality is reduced afterwards. *FastText* (Bojanowski et al., 2017) is a more recent method based on word2vec, but taking into account sub-word, character-level information. Instead of learning a representation for a word, fastText learns a representation for all character n-grams in a word, thereby jointly optimizing semantic representations of character n-grams and full words. FastText performs on par with state-of-the-art deep learning models on various text classification tasks, at a fraction of the computational cost in both the training and inference phase (Joulin et al., 2017). This indicates the power of incorporating character-level information in training.

2.2.3 *Sentence embeddings*

The problem of composing sentence representations from word representations is an active field of study. A simple method would be element-wise vector addition or multiplication (Mitchell and Lapata, 2010). As Mikolov et al. (2013b) also suggest, vector addition seems to represent some kind of semantics for a sentence. Additive sentence embeddings represent a robust, semantically aware sentence embedding baseline. IDF-reweighted additive

sentence embeddings usually perform even better: in this case, every word embedding is divided by the number of sentences it occurs in before all words are summed.[5] From a linguistics perspective, however, sentence composition is a complex matter that probably requires more expressive models in order to learn the meaning of word combinations. Various methods for learning sentence embeddings have been proposed.

Paragraph Vector (Le and Mikolov, 2014) provides an unsupervised, word2vec-like method that predicts vectors for pieces of text (e.g. sentences, paragraphs, or documents) by predicting the words they contain[6]. For various tasks, such as similarity/analogy tasks on document level and document clustering, Paragraph Vector provides an efficient way of embedding. For sentence similarity tasks, however, there seem to be more successful methods.

Auto-encoders are an intuitive approach towards semantic representations of sentences: using some neural network architecture, auto-encoders are trained unsupervisedly on reproducing the input sentence as output through a bottleneck vector. This vector could be a robust and accurate representation of the semantic and syntactic information of a sentence. Hill et al. (2016) proposed a sequential denoising auto-encoder (*SDAE*), a recurrent neural network (RNN) architecture to compress a corrupted (noisy) sequence of words in word order by training it to reproduce the original, denoised, sentence. Besides sequence-based RNN approaches, more auto-encoder architectures have been proposed for sentence representation. Socher et al. (2011) proposed an unfolding recursive auto-encoder (*RAE*), which is guided by a binary syntactic tree of the sentence. It recursively combines (i.e. folds) word vectors, thereby ultimately encoding the whole sentence. Then it decodes (i.e. unfolds) the sentence vector using the same tree, thereby training the folding and unfolding operation. Unfolding RAEs outperformed the former state-of-the-art in paraphrase detection. Kågebäck et al. (2014), however, found that word vector addition worked better in practice for MDS, although this could be the result of using a dataset consisting of low quality text[7].

Another encoder-decoder, but non-auto-encoder model used to produce sentence embeddings is the *Skip Thought* model (Kiros et al., 2015). This

5 We will use this embedding technique in Chapter 4.

6 This concerns the distributed bag-of-words model (PV-DBOW), which is more efficient during training and more popular for down-stream applications than the distributed memory model (PV-DM).

7 The Opinosis dataset (Ganesan et al., 2010), respectively.

model encodes sentences using an RNN and is trained on predicting the words in its surrounding sentences. In this sense, the approach exploits a sentence-level Distributed Hypothesis: it constructs a distribution over the words in the surrounding sentences given the current sentence. As both the encoder and decoder consist of an LSTM network with a relatively large number of hidden units, training and inference are slow.

FastSent (Hill et al., 2016) uses the same distributive sentence-level hypothesis as Kiros et al. (2015), but proposes a much simpler additive model to compose a sentence vector from its word vectors. The word vectors are then trained with the objective to, when added together for a sentence, best predict the separate words in the surrounding sentences by producing a softmax over the whole vocabulary.

A slightly different approach compared to FastSent is *Siamese CBOW* (Kenter et al., 2016). Despite using the same signal, in this model, the word vectors are averaged instead of summed. More importantly, instead of training word vectors with the objective to predict the separate words in the surrounding sentences, it minimizes the cosine similarity between the averaged sentence vectors. Siamese CBOW and FastSent reach comparable accuracies on sentence similarity tasks.

Pagliardini et al. (2017) propose a method based on the conceptual idea of fastText which they call *Sent2Vec*. Instead of jointly optimizing words and their character n-grams to predict surrounding words in the sentence, a sentence's word n-grams (unigrams only, or unigrams + bigrams) are jointly optimized to predict its words. This small adaptation surprisingly yields promising results; it reaches state-of-the-art results on many benchmark sentence embedding tasks. As the model is trained on another text corpus however (i.e. the full English Wikipedia, as opposed to the Toronto Book Corpus for FastSent and Siamese CBOW), it is hard to fairly compare performance of this system.

Arora et al. (2017) propose a different, surprisingly simple unsupervised approach for sentence embedding construction called the *smooth inverse frequency (SIF)* embedding, which they propose as a new baseline for sentence embeddings. Their approach is summarized in Algorithm 1.
Instead of using a neural approach, they start by computing a mean of all words in the sentence, weighting every word based on its estimated probability (i.e. normalized frequency in some corpus), usually setting parameter $a = 10^3$. This should be roughly comparable to the previously discussed

Algorithm 1 SIF embedding algorithm by Arora et al. (2017)

Input: Word embeddings $\{v_w : w \in \mathcal{V}\}$, a set of sentences (bag-of-words) $\mathcal{S}$, parameter a, and estimated probabilities $\{p(w) : w \in \mathcal{V}\}$ of the words.

Output: Sentence embeddings $\{v_s : s \in \mathcal{S}\}$.

1: **for all** sentences $s \in \mathcal{S}$ **do**

2: $\quad v_s \leftarrow \frac{1}{|s|} \sum_{w \in s} \frac{a}{a + p(w)} v_w$

3: **end for**

4: Compute the first principal component u of $\{v_s : s \in \mathcal{S}\}$

5: **for all** sentences $s \in \mathcal{S}$ **do**

6: $\quad v_s \leftarrow v_s - uu^\top v_s$

7: **end for**

IDF-reweighting. After computing all means, the first principal component is computed, and the projection of all sentence vectors on this first principal component is removed. The authors claim that this common component removal reduces the amount of syntactic information contained by the sentence embeddings, thereby allowing the semantic information to be more dominant in the vector's direction.

In this thesis, not all the aforementioned sentence embedding methods will be used; we limit ourselves to the use of simple methods such as (IDF-reweighted) averages/sums of word vectors and the SIF embedding.

2.3 SENTENCE SELECTION

The general objective for sentence selection is to select maximally informative sentences, without information overlap between the sentences, to maximize the coverage of the original article by the summary. Although the number of sentence selection methods is too large to provide a full overview, some well-known methods which are commonly used as baselines are discussed below.

2.3.1 *Maximal marginal relevance*

A well-known, classic, unsupervised algorithm used for sentence selection is the Maximal Marginal Relevance (*MMR*) algorithm (Carbonell and Goldstein, 1998). MMR is an algorithm that balances the trade-off between relevance (or informativeness) and coverage, thereby incorporating diversity into the

summary. Using a greedy approach selecting one sentence every iteration, it picks the sentence from the article that is both relevant and dissimilar from already picked sentences until a word maximum is reached. It is an efficient method, as it only depends on the already picked sentences and the candidate sentence's own informativeness in every iteration.

2.3.2 *Centroid-based: MEAD*

Another selection method coined *MEAD* (Radev et al., 2004) is a centroid-based algorithm. It assumes a semantic vector representation of every sentence (TF-IDF in the original paper). It then clusters the sentences and computes centroids for each cluster. Combining centroid-sentence cosine similarities and other sentence features (i.e. length)[8] to score all sentences in each cluster, a subset of sentences of each cluster is then selected to form a summary.

2.3.3 *Graph-based: TextRank & LexRank*

A popular class of sentence selection methods are graph-based methods. Graph-based methods are usually based on Google's famous PageRank algorithm (Page et al., 1999), and are generally considered more flexible; while centroid-based methods define a hard clustering on all sentences, the graph-based approach allows sentences to be more or less connected to each other. Both *TextRank* (Mihalcea and Tarau, 2004) and *LexRank* (Erkan and Radev, 2004) construct a graph, where the vertices represent sentences. They use some sentence similarity measure[9] to construct (weighted) edges between sentences. Important sentences are expected to be central in the graph, hence they are often being 'recommended' by other, similar sentences. By iteratively scoring sentence importance by recommending edges in the graph, the most important sentences attain higher scores at convergence. When normalizing the edge weights of the graph to sum to 1, the iterative process converges to the stationary distribution of the Markov chain of sentences. The vertices with the highest probability represent the most important sen-

8 Although the weights for these parameters could be learned, Radev et al. chose to assign equal weights instead.

9 Normalized word overlap for TextRank, and TF-IDF cosine similarities for LexRank.

tences, and are selected for the summary. To assure diversity, LexRank re-ranks the sentences using an MMR-related method before selection.

2.3.4 *Greedy submodular optimization*

Lin and Bilmes (2010, 2011) show that both informativeness and diversity mentioned above are naturally modelled as monotone submodular functions. Monotone submodular functions mathematically capture the sense of *diminishing returns* (i.e. the fact that the added value of an element to some set is bigger than for its proper supersets). This naturally models the selection of summary sentences, as the added value of a sentence should decrease when the summary length increases. Maximizing monotone submodular functions is unfortunately an NP-complete problem, but the greedy algorithm (i.e. Algorithm 2) proposed in the 2010 paper approximates the maximum with good guarantees.[10]

Algorithm 2 Greedy algorithm by Lin and Bilmes (2010)

Input: The set of all article sentences V, the scaling factor r, the budget constraint B, and the monotone submodular function $f(\cdot)$.

Output: The set of summary sentences G_f.

$G \leftarrow \emptyset$ ▷ initialize the (initially empty) set of summary sentences G
$U \leftarrow V$ ▷ initialize the set of candidate summary sentences U as V
while $U \neq \emptyset$ **do**
  $k \leftarrow argmax_{l \in U} \frac{f(G \cup \{l\}) - f(G)}{(length_l)^r}$
  if $\sum_{i \in G} [length_i] + length_k \leq B$ **and** $f(G \cup \{l\}) - f(G) \geq 0$ **then**
    $G \leftarrow G \cup \{k\}$
  end if
  $U \leftarrow U \setminus \{k\}$
end while
$v^* \leftarrow argmax_{v \in V, length_v \leq B} f(\{v\})$
return $G_f = argmax_{S \in \{\{v^*\}, G\}} f(S)$

The scaling factor r punishes longer sentences; r is usually set to 0.3. The summary budget constraint B is usually set to 100 words. The algorithm considers the sentence with the biggest added value to the summary at every time step, and includes it in the summary if the added value is positive, and if

10 The value of the objective function maximized by the greedy approach is within $\frac{1-e}{e} \approx 0.63$ of the maximum value of the objective function for the problem.

it does not violate the budget constraint. When all sentences are considered, the algorithm returns either the resulting summary, or the best singleton summary if its value surpasses the resulting summary's value.

A mixture of various monotone submodular functions is also monotone submodular, hence a mixture of various submodular objective functions can also be maximized with guarantees using Algorithm 2. Two submodular functions that together model informativeness and diversity can thus be greedily maximized using the algorithm. In fact, Lin and Bilmes (2010) showed that MMR (as mentioned above) is naturally modelled within their submodular framework. In 2011, they proposed different objective functions for modelling informativeness and diversity, surpassing state-of-the-art results on various multi-document summarization datasets.

2.3.5 *Supervised sentence classifiers: recurrent neural networks*

Supervised approaches towards extractive summarization are rare. A recent, very successful approach was proposed by Cheng and Lapata (2016). They propose a deep learning approach by utilizing a recurrent neural network (RNN) architecture to 'read' all sentences of a document, combined with a single-layer convolutional neural network (CNN) to learn to encode words into sentences. Although the authors propose both an abstractive and an extractive approach, we will only consider the extractive approach given the scope of the current research.

Cheng and Lapata essentially treat extractive summarization as a sentence labeling task using an RNN architecture to both encode the document and label the salience of every sentence. Sentence importance is learned from a gold standard of hundreds of thousands of sentence-labeled news articles, based on pairs of news articles and bullet point summaries from CNN and DailyMail. This approach yields good results on the dataset itself, but also appears to generalize well to the DUC-2002 dataset. For a more thorough discussion on recurrent neural networks and their use for extractive summarization, see Chapter 5.

2.4 EVALUATION: ROUGE

ROUGE (Lin, 2004) is the most commonly used summary evaluation metric. It attempts to evaluate the correspondence between an automatically gener-

ated summary and one or more human reference summaries. ROUGE was originally only recall-oriented: it measured what fraction of a reference summary was captured by the system summary and thus how well the reference summary was 'covered'.

Determining what fraction of a reference summary is captured is not straightforward, however. This is solved by ROUGE-1 and ROUGE-2, considering either unigrams or bigrams as the 'atoms' of the summary, respectively. The original ROUGE-N metric is hence based on recall-oriented n-gram co-occurrence. For a set of reference summaries $\mathcal{S}$ and a system summary C, the ROUGE-N score is defined as

$$\text{ROUGE}_n(\mathcal{S}, C) = \frac{\sum_{S \in \mathcal{S}} \sum_{gram_n \in S} Count_{match}(gram_n)}{\sum_{S \in \mathcal{S}} \sum_{gram_n \in S} Count(gram_n)},$$

where n indicates the size of the n-gram, and $Count_{match}(gram_n)$ is the number of n-grams of the reference summary S covered by the system summary C. The metric divides the number of n-grams matching a reference summary by the total number of n-grams in the reference summary, and averages this over all reference summaries, thereby computing the coverage of the reference summaries by the generated summary.

In more recent versions of ROUGE, precision is also taken into account, as a summary containing irrelevant information is undesirable. This means that the percentage of the system summary which is also in the reference summary influences the summary evaluation. In most cases, the harmonic mean between the precision and recall is considered the fairest metric and is called the F-score. We will use either recall or F-score for summary evaluation, depending on the dataset.

Both ROUGE-1 and ROUGE-2 are reported in all system summary evaluations against human-written summaries throughout this thesis. ROUGE-1 is generally considered the most important metric, as it is known to correlate most strongly with human summary evaluation. It is also less biased by grammatically similar sentences. We therefore prefer ROUGE-1 for its correlation with human evaluation. We do not use ROUGE's built-in stemming or stop word removal options, in order to keep the evaluation results transparent and simple.

	ROUGE-1	ROUGE-2
Mean	48.84	21.84
SD	11.07	13.03

Table 1: The means and standard deviations ROUGE F-scores between reference summaries on the DUC-2002 dataset, for all articles with two reference summaries.

2.4.1 ROUGE *shortcomings*

Although ROUGE is the most common summary evaluation method, it is far from perfect. Note that the goal of summary evaluation is to measure to what extent system summaries and (sets of) reference summaries contain the same information. In order to approximate this, ROUGE is a function of N-gram overlap. The same information can be captured by different word combinations, however. In this light, ROUGE does not necessarily evaluate the information coverage of a system summary, but merely the word coverage.

This weakness becomes clear when we compute the average ROUGE F-scores between reference summaries of the same article. Ideally, ROUGE scores between two reference summaries would be close to 100%, as the meanings conveyed by both summaries should be similar. We compute ROUGE scores between reference summaries of the DUC-2002 dataset (described in Section 3.2) for articles with two available reference summaries.[11] Although Table 1 might also be influenced by the purely semantic difference between reference summaries due to the inter-rater agreement, the fact that the ROUGE-1 score does not pass 50% is alarming. As it is the best measure we have, however, we will still use it for all quantitative summary evaluations throughout this thesis.

11 This is the case for 550 of the 567 articles; for 17 articles, there is only one reference summary available.

3

STUDY 1: A SURVEY OF EXISTING EMBEDDING-BASED SUMMARIZATION METHODS

Although word embeddings are central to state-of-the-art models in almost every subfield of NLP research, this trend has not fully reached the field of extractive summarization yet. There have been some efforts to incorporate word or sentence embeddings in extractive summarization methods, but every paper aims at high performance on a different dataset, or uses a slightly different evaluation measure. This makes comparison between these efforts difficult. In this first study, we attempt to make an honest, thorough comparison between various embedding-based, extractive summarization approaches against some robust, commonly used baselines. In order to achieve this, we implemented all reported algorithms. We made all implementations available as an open-source project on GitHub.[1] By evaluating all algorithms in the same way, we aim to make the comparison as fair as possible.

The considered embedding-based extractive summarization methods are aimed at the task of multi-document summarization (MDS). Although our problem is not about document sets, but about single articles on Blendle's platform, we still consider some multi-document summarization datasets for the comparison in this chapter, as the existing embedding-based algorithms were designed for this task. Besides, the MDS task gives a better insight in the ability of a summarization system to pick diverse sentences, as many similar sentences might be available in a cluster of related articles. We will consider two MDS datasets, which are both commonly used for MDS evaluation[2]: the TAC-2008 and the Opinosis dataset. For single document evaluation, we will use the DUC-2002 dataset. All datasets will be discussed in Section 3.2.

To our knowledge, embedding-based extractive summarization methods have never been compared to strong baselines or each other. Although the reproduction and comparison of earlier results is often overlooked, it has proven to be very informative before, e.g. in the fields of neural language modelling

1 https://github.com/blendle/research-summarization

2 Note that the embedding-based summarization methods under consideration also use one of these datasets for evaluation.

(Melis et al., 2017) and information retrieval (Armstrong et al., 2009). In this study, we attempt to provide a comparable review for embedding-based extractive summarization.

Before arriving at the discussion of datasets and the final comparison, however, the different embedding-based methods and commonly used baselines that we consider are first explained. Although the elements of each method were already roughly explained in Chapter 2, the combination of elements that each paper uses is described in more detail in the next section. TextRank, a commonly used SDS baseline, is not described in this chapter; for an elaboration on TextRank, see Chapter 4.

3.1 EMBEDDING-BASED SUMMARIZATION METHODS & COMMON BASELINES

3.1.1 *The beginning-of-article baseline: LEAD*

The most straightforward baseline considered in this thesis is the *LEAD* baseline. This baseline simply selects the first n sentences of an article for a summary, given a length constraint; it ranks the sentences purely on their order. Although this baseline is only applicable to SDS tasks,[3] it is an undervalued baseline in the field of extractive summarization, as we will see later in this chapter.

3.1.2 *Lin & Bilmes (2010, 2011): greedy submodular baselines*

As discussed in Section 2.3.4, Lin and Bilmes (2010) proposed a greedy algorithm to optimize a (mixture of) submodular objective functions. In their original paper, they proposed a mixture of two submodular objective functions, closely resembling the classical MMR approach (Carbonell and Goldstein, 1998):

$$f^{obj}(C) = \sum_{i \in D \setminus C} \sum_{j \in C} \text{sim}(i,j) - \lambda \sum_{i,j \in C; i \neq j} \text{sim}(i,j),$$

where λ is set to 4, $\text{sim}(\cdot,\cdot)$ corresponds to the cosine similarity between the TF-IDF vectors, and D and C are the set of all sentences in the document

3 The *LEAD* baseline does not make sense for clusters of articles, given its sequential nature.

and the set of summary sentences, respectively. This objective function thus measures coverage by summing all sentence similarities between summary sentences and the other sentences (also known as the graph cut function), and punishes redundance by summing the within-summary sentence similarities.

Using this mixture as the objective for the greedy algorithm surpassed state-of-the-art results on multiple MDS datasets. In 2011, Lin and Bilmes proposed another mixture of two functions, again combining measures for coverage and diversity, and again surpassing the state-of-the-art on various MDS datasets.[4] They use the following coverage and diversity functions:

$$f^{coverage}(C) = \sum_{i \in D} \min \left\{ \sum_{j \in C} \text{sim}(i,j),\ \alpha \sum_{k \in D} \text{sim}(i,k) \right\}$$

$$f^{diversity}(C) = \sum_{k=1}^{K} \sqrt{\sum_{j \in C \cap P_k} \frac{1}{N} \sum_{i \in D} \text{sim}(i,j)}$$

$$f^{obj}(C) = f^{coverage}(C) + \lambda f^{diversity}(C),$$

where the threshold value $\alpha = \frac{5}{N}$, the number of sentence clusters[5] $K = 0.2N$, P is the set of K-means clusters, and the weighting term $\lambda = 6$.

As for the coverage function, we see that the previously used graph cut function is replaced by a function that takes a minimum for every sentence in the document. For every document sentence, the minimum is selected from two values: the similarities between the summary sentences and the current sentence, and the similarities between all sentences and the current sentence. Since all similarities are positive, as long as $\alpha = 1$, the second term is always at least as large as the first term, since $C \subseteq D$. By choosing $\alpha < 1$, the second term provides a threshold for the first term: if the current document sentence is already represented in the summary well enough, it is 'saturated', and does not influence the score of newly added sentences. This threshold thus already ensures that the selection of summary sentences is not dominated by one cluster of highly similar sentences.

4 Specifically, on the DUC 2004-2007 MDS datasets.

5 In other words, every cluster contains 5 sentences on average; the clustering is a K-means clustering based on TF-IDF vectors.

The diversity function is now cluster-based. The function returns the sum of the square root of the average similarity of all summary sentences per cluster. Due to the square root function, the added value of a second summary sentence from the same sentence cluster is naturally smaller, modelling the diminishing returns property of summarization.

3.1.3 *Kågebäck et al. (2014): enriching Lin & Bilmes (2011)*

Although the algorithm of Lin and Bilmes (2011) performs well, it does not benefit from the information word embeddings might add to the model. Kågebäck et al. (2014) experimented with their summarization method, using different word embedding-based sentence embeddings instead of TF-IDF vectors. The first, proposed sentence embedding is constructed using an unfolding recursive auto-encoder (*uRAE*), based on Socher et al. (2011). This recursive (not to be confused with recurrent) neural network architecture learns to 'collapse' word embeddings into a sentence embedding in an unsupervised manner, by essentially learning the composition of words in a binary syntactic tree. Kågebäck et al. also find that simply using additive sentence embeddings yields better results on the Opinosis dataset (which is discussed in Section 3.2) than using TF-IDF vectors or uRAE-based sentence embeddings, however. We chose to reproduce their additive approach.

3.1.4 *Kobayashi et al. (2015): proposing alternative objective functions*

In the same framework proposed by Lin and Bilmes, Kobayashi et al. (2015) take a somewhat more radical approach than Kågebäck et al. (2014). They propose two entirely different objective functions: DocEmb and EmbDist.

DocEmb uses a slightly different angle in representing text; it simply sums all words in a text, irrespective of sentence boundaries. The basic component of the algorithm is a document embedding of the entire document. This embedding is the sum of all word embeddings in the document. Using the greedy algorithm of Lin and Bilmes, a similar additive embedding is computed for every candidate summary, taking cosine similarity between the document and summary embedding as the objective function:

$$f^{obj}(C) = \frac{v_C \cdot v_D}{\|v_C\| \, \|v_D\|},$$

where $\|v_C\|$ and $\|v_D\|$ are the summary and document vector, respectively. This gives an intuitive notion of what a summary should be: the best summary is the one that is most similar to the original text.

Formally, however, DocEmb does not fit in the algorithm of Lin and Bilmes, as this objective function is not submodular. The second objective function, EmbDist, solves this issue. EmbDist is somewhat more complex: instead of maximizing the distance between a document and summary vector, the inverse average distance between every document sentence and its closest summary sentence is now maximized[6]:

$$f^{obj}(C) = -\sum_{s \in D} N(s, C),$$

where $N(s, C)$ indicates the distance between a document sentence and the most similar sentence in the set of summary sentences C (excluding the document sentence itself). The authors also tried this approach on word level (i.e. minimizing the average distance between every word in the document and the word's closest summary word), but did not report the results. In our final comparison, we report the results on both sentence and word level.

3.1.5 *Yogatama et al. (2015): maximizing semantic volume*

Instead of maximizing the coverage of the original text, while controlling the redundance, Yogatama et al. (2015) propose another maximization objective. The authors choose to maximize semantic volume: they select the subset of sentences of which the convex hull in semantic sentence space is maximal. This idea is illustrated in Figure 6. As this is NP-hard, Yogatama et al. use a greedy hill-climbing algorithm to approximate maximal semantic volume instead. As for sentence embeddings, the authors use singular value decomposition on sentence bigram counts to obtain 600-dimensional sentence embeddings; sentence embeddings are thus not built up from trained word embeddings in this case.

We have not succeeded in reproducing the results of this paper; our — to our knowledge — exact reproduction performs significantly worse than

6 Besides directly using the distances, they also proposed to transform the distances in EmbDist to exponential or log scale. As we did not see any significant effect of this transformation, we implemented EmbDist without transformation.

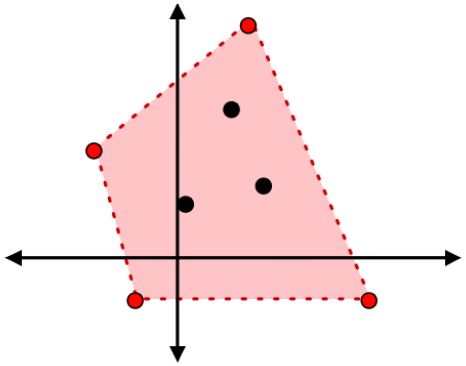

Figure 6: The conceptual idea of maximizing the semantic volume, simplified to a two-dimensional space. Sentences are represented as dots in the 2-D vector space. Given some length restriction, the optimal solution for these 7 sentences is the set indicated in red, spanning the largest semantic volume. Adopted from Yogatama et al. (2015).

originally reported. We tried to contact the authors, but have not managed to obtain the original code of the paper.

3.2 DATASETS & WORD2VEC MODELS

As mentioned above, we use three different datasets for the comparison, in order to get a more generalized sense of the performance of all algorithms considered in this chapter. We consider both MDS and SDS datasets. All of them are commonly used for summarization evaluation. We consider DUC-2002, TAC-2008 and Opinosis.

3.2.1 *DUC-2002*

DUC-2002 is a dataset that is commonly used for SDS. The dataset contains a total of 567 news articles. All news articles are from respected American news sources (e.g. Wall Street Journal, Financial Times). For every DUC document, 1 (for 17 articles) or 2 (for 550 articles) human-written reference summaries are available. The goal is to construct a summary with a maximum of 100 words. For evaluation, gold-standard summaries of $\sim$100 words are included for each article. The average ROUGE-N F-score over the reference summaries is the evaluation metric. As opposed to the other datasets, DUC-

2002 comes with a version where sentences were already splitted, and metadata (i.e. author, document ID, newspaper etc.) is already removed from the content. We used the 'cleaner' version of the dataset throughout this thesis, as this sanitized content is closer to Blendle's own data.

3.2.2 *TAC-2008*

TAC-2008, on the other hand, is a well-known MDS dataset. The TAC-2008 dataset contains 48 clusters of 20 news articles each. Every cluster contains articles on the same topic. Each of these clusters is separated into two subclusters of 10 articles, where the second cluster provides a news update over the first one. Only the non-update proportion of every cluster is considered for the current comparison. For every document cluster, 4 human-written, 100-word summaries were provided. As with DUC-2002, average ROUGE-N F-scores are used to compute system summary quality.

3.2.3 *Opinosis*

The Opinosis dataset is a another kind of MDS dataset: it contains Amazon product reviews instead of news articles. Although this is less relevant to Blendle, it still gives an insight in the generalizability of the algorithms. It contains 51 clusters of user reviews, where each cluster contains reviews for a specific product. Every cluster is relatively small: it contains between 50-575 sentences. For evaluation, every topic is accompanied with 4-5 short reference summaries of about 3 sentences. This is significantly less than 100 words, which is why F-scores would be severely biased by the lack of precision when producing 100-word summaries. This is why we chose to report recall scores for this dataset instead.

3.2.4 *Word2vec models*

As the foundation for sentences throughout this whole thesis, two different word2vec models were considered. Both word2vec models are trained using the skip-gram architecture with negative sampling, as explained in Section 2.2.

The first model is arguably the most common pre-trained vector model in NLP literature: the Google News vector model[7]. This model is trained on a part of Google's News dataset, which contains about 100 billion words. It contains 300-dimensional vectors for 3M words and phrases. Only the model is public; the training data has not been published. The second model we consider is trained on Blendle's own news dataset. It contains 300-dimensional vectors of the ∼0.2M most frequent lower-cased words from the news corpus, without punctuation. Although the training corpus is significantly smaller (∼277M words), the data only contains articles from high-quality US newspapers. We trained this word2vec model using Gensim.[8] As we will see later, this results in a well-performing, small vector model.

3.3 EXPERIMENT

As mentioned above, we tried to make the comparison as honest as possible. As for the experimental conditions, this means:

- Every summarization method was allowed to select exactly enough sentences to reach (and usually slightly overshoot) the 100 word limit;
- We used the same ROUGE settings for all different methods per dataset;[9]
- We report the results of every embedding-based model with both Google's and Blendle's news vectors;
- All approaches using word2vec-based sentence embeddings are implemented using IDF-reweighted word embedding sums,[10] to make the performance comparison between methods in this chapter not too dependent on the exact form of a sentence embedding. This means that every word embedding was first divided by the number of sen-

7 https://code.google.com/archive/p/word2vec/

8 https://radimrehurek.com/gensim/models/word2vec.html; we used the skip-gram model with negative sampling, and trained it for 20 epochs. Minimum token count was set to 10, the number of negative samples to 5, and the window size to 5.

9 We did not use the stemming or stop word removal option during ROUGE evaluation, and considered only the first 100 words of every summary for evaluation.

10 Note that the IDF-reweighting is smooth: we add 1 to all document frequencies, as if there is an extra document containing all words.

tences of the article (cluster) it occurs in, before word embeddings were summed;[11]

- For the DUC-2002 and TAC-2008 datasets, we report F-scores, as these are preferable for measuring performance (see Section 2.4). For the Opinosis dataset, we report recall scores, as the 100-word system summaries are usually longer than the reference summaries, resulting in distorted F-scores due to variation in precision;

- In all evaluation tables throughout this thesis, we will report the best score and all scores that are not significantly different from the best score in boldface. In all cases, the difference between summarization methods on ROUGE measures is tested with a paired-samples permutation test (Nichols and Holmes, 2002), randomly sampling 100,000 permutations for each test, as we cannot assume a normal distribution of ROUGE scores over all articles. We did *not* use the Wilcoxon signed-rank test (Wilcoxon, 1945), as this test is known to underestimate the significance of differences (Sanderson and Zobel, 2005; Smucker et al., 2007). We assume scores to be significantly different if $p < 0.05$, where we correct for the number of tests using the Holm-Bonferroni method (Holm, 1979).

The performance of the following summarization systems are evaluated and compared:

- Baselines:
  - The *LEAD* baseline;
  - LB-2010: the common MDS baseline by Lin and Bilmes (2010);
  - LB-2011: another common MDS baseline by Lin and Bilmes (2011);
  - TextRank: a common SDS baseline by Mihalcea and Tarau (2004), discussed in more detail in Chapter 4.

- Embedding-based methods:
  - EmbDist (sent): the EmbDist algorithm on sentence level, as used by Kobayashi et al. (2015);

11 This means that our implementation of method by Kågebäck et al. (2014) uses a slightly more sophisticated sentence embedding, as we perform IDF-reweighting on the additive sentence embedding used in the original paper.

- EmbDist (word): the EmbDist algorithm on word level, as originally proposed (but not evaluated) by Kobayashi et al. (2015);
- DocEmb: the DocEmb algorithm by Kobayashi et al. (2015);
- Kågebäck: the algorithm by Kågebäck et al. (2014);
- LB-2010 (w2v): the submodular MMR-like summarization method of Lin and Bilmes (2010), using IDF-reweighted additive sentence embeddings instead of TF-IDF vectors.

3.4 RESULTS & DISCUSSION

The results of the comparison are shown in Table 2. We disregard the Opinosis dataset for the rest of our analysis, as the large variance in paired differences and the small number of document clusters causes the statistical power to be too low: statistically, almost all methods perform equally well on this dataset. We thus have one MDS and one SDS dataset to compare the various methods.

Considering the TAC-2008 dataset, the best performance is achieved by the submodular MMR method of Lin and Bilmes (2010), with comparable results for TextRank, the method of Lin and Bilmes (2011), and the EmbDist method on sentence level by Kobayashi et al. (2015). EmbDist only achieves top performance with Blendle's word2vec model. MSV (Yogatama et al., 2015) performed much worse than reported in the original paper; the first author did not respond to our request for the original research code when we emailed about our findings.

On the DUC-2002 dataset, *LEAD* clearly achieves top performance. If *LEAD* would not be considered, TextRank achieves the best result. TextRank's ROUGE-1 performance is matched by DocEmb using either Google's or Blendle's word2vec model. In general, we draw the following conclusions based on the results.

Embedding-based methods do not necessarily perform well. The considered non-word2vec baselines tend to outperform the embedding-based methods overall. Moreover, the submodular MMR method of Lin and Bilmes (2010) with word2vec embeddings instead of TF-IDF vectors is outperformed by all other methods. This is interesting, as we apparently do not only add free information, but also a lot of noise that distracts the selection method.

		DUC-2002		TAC-2008		Opinosis	
		ROUGE-1	ROUGE-2	ROUGE-1	ROUGE-2	ROUGE-1	ROUGE-2
Google w2v	EmbDist (sent)	41.18	18.05	32.12	6.58	49.95	12.91
	EmbDist (word)	42.17	17.65	28.29	4.78	43.28	7.40
	DocEmb	*43.85*	18.86	31.09	5.79	**53.42**	**14.98**
	Kågebäck	42.49	17.97	29.87	5.48	**50.78**	**14.49**
	LB-2010 (w2v)	27.88	9.35	29.90	5.85	51.01	**14.70**
Blendle w2v	EmbDist (sent)	40.99	18.26	**33.13**	**7.80**	**49.87**	11.88
	EmbDist (word)	42.12	17.48	29.84	5.80	41.92	7.33
	DocEmb	*43.70*	18.65	30.89	5.92	**51.28**	**12.19**
	Kågebäck	41.97	17.73	28.25	4.37	**51.09**	**15.80**
	LB-2010 (w2v)	25.85	8.31	29.05	5.21	51.44	**13.36**
non-w2v	LB-2010	38.28	14.18	**34.48**	**7.82**	**55.57**	**18.00**
	LB-2011	43.53	18.66	**32.27**	**6.79**	**53.01**	**18.12**
	TextRank	44.49	20.25	**33.50**	**8.53**	49.13	**14.60**
	MSV	40.58	16.58	26.66	3.48	**51.47**	**13.89**
	LEAD	**46.00**	**23.01**	—	—	—	—

Table 2: Rouge scores on the Opinosis, TAC-2008, and DUC-2002 datasets for various algorithms. Results for both Google's word2vec model and the word2vec model trained on Blendle's data are reported. Note that the scores for TAC-2008 and DUC-2002 are Rouge F-measures, while the scores on Opinosis are Rouge R-measures. For DUC-2002, the best scores behind *LEAD* are italicized.

It seems difficult to cope with this noise during sentence selection.

TF-IDF-based methods seem to perform well on MDS. This could be explained by the fact that more sentences are available per problem in the MDS setting. As the number of sentences in a problem grows, there are more examples of which words are shared by sentences, and which words are unimportant due to their frequency. This results in a more accurate indication of relations between sentences, as the inverse document frequencies become more reliable. This also implies that the added information of word2vec, if interpreted effectively, could especially help in an SDS setting, as TF-IDF is less effective in that case due to the smaller number of sentences per problem.

In general, TextRank appears to be a very robust baseline. It achieves top performance on the MDS dataset, and is the best SDS method if we disregard the *LEAD* baseline for now. As TextRank does consider diversity during sentence selection, it seems that relevance is the more important factor. TextRank thus seems a robust unsupervised relevance ranker.

The Blendle and Google word2vec model achieve similar performance. This is an interesting finding, as the Blendle model is trained on much less data. The Blendle model is also much smaller in terms of vocabulary. Although the corpus Google used is not public, we hypothesize that the Blendle data is much cleaner, and the text is of higher quality. This makes sense, as the English Blendle corpus consists of only high-quality US newspaper articles.

***LEAD* is an undervalued SDS baseline.** In other research, the *LEAD* baseline is either outperformed, unmentioned, or even weakened.[12] We unexpectedly found, however, that the baseline consistently outperforms other methods on the DUC-2002 dataset. This makes sense: news articles tend to start with important information. Although the fact that *LEAD* outperforms the other methods is an unexpected finding, it is also an important one: it indicates that the sequential property of text should not be disregarded. The use of sequential information and the challenge of outperforming *LEAD* are the subject of Chapter 5.

While in theory embeddings only add information to the model for free by unsupervisedly learning word meaning, it seems difficult to use this information to improve summary quality. In order to improve our understanding of using embeddings for extractive summarization, our next step is to look into enriching the well-performing[13] TextRank algorithm with sentence embeddings. In the next chapter, we will hence study the effect of sentence embeddings on TextRank.

12 Some papers interpret the *LEAD* baseline as the first 3 sentences of every article (e.g. Cheng and Lapata (2016)), which is an unfair disadvantage.

13 Although *LEAD* achieves better performance, this baseline is not easily integrated with sentence embeddings. Improving over *LEAD* will be the subject of Chapter 5.

4

STUDY 2: ENRICHING TEXTRANK WITH WORD EMBEDDINGS

In this second study, we further investigate the added value of using word2vec-based sentence embeddings for extractive summarization. Instead of evaluating existing embedding-based methods, as done in Chapter 3, we will investigate the added value of substitution of sentence embeddings into TextRank, the best-performing SDS method behind the *LEAD* baseline. We will first elaborate on the TextRank algorithm in Section 4.1. We will then discuss the reproduction of TextRank's original results in Section 4.2[1], the substitution experiment in Section 4.3, and conclude the chapter with a discussion of the results in Section 4.4.

4.1 THE TEXTRANK ALGORITHM

The TextRank algorithm (Mihalcea and Tarau, 2004), as shortly described in Chapter 2, is a graph-based extractive summarization method achieving good performance on the DUC-2002 single document summarization dataset. It is a common SDS baseline. It is based on Google's PageRank algorithm (Page et al., 1999), which is visually represented in Figure 7. In the TextRank case, the sentences in the text are the vertices in the graph. The weights of the edges in the graph are defined as the similarity of its vertices (i.e. sentences). The similarity measure used for TextRank is a fairly simple word co-occurrence measure. Given two sentences S_i and S_j, where a sentence is defined as a set of the N words it contains: $S_i = \{w_1^i, w_2^i, \ldots, w_N^i\}$, similarity between the two sentences is defined as length-normalized word overlap:

$$\text{similarity}(S_i, S_j) = \frac{S_i \cap S_j}{\log(|S_i|) + \log(|S_j|)}$$

1 Note that we used our final reproduction version for the comparison in Chapter 3; we describe the complications that we encountered during our reproduction of TextRank in this chapter.

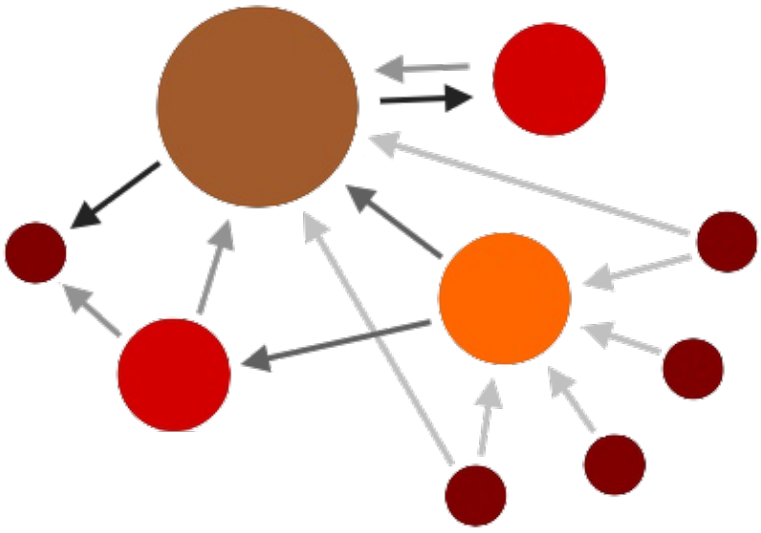

Figure 7: Static visual representation of the PageRank algorithm. The more arrows are pointing to a node, the more important (i.e. bigger) it becomes. The color of an arrow indicates its weight, which corresponds to the node size. Note that in the TextRank case, all relations are symmetric.

The fully connected graph is initialized with the similarity-based weighted edges, and initial vertex scores of 1. The vertices are then re-scored by taking into account the scores of neighboring vertices, weighted by the edge weight. The new score $S(V_i)$ of vertex i becomes:

$$S(V_i) = (1-d) + d \times \sum_{V_j \in In(V_i)} \frac{w_{ji}}{\sum_{V_k \in Out(V_j)} w_{jk}} S(V_j),$$

where d is a constant damping factor, usually set to 0.85. In our case, the indegree and outdegree of every node are always equal, as the similarity measure is symmetric. This means that sentences 'recommend' each other based on their similarity and their current scores. Graph re-scoring is repeated until the scores in the graph converge[2], which is usually after 20–30 iterations. The algorithm then returns a sentence ranking (highest-scored sentences first), of which the highest-ranked sentences are picked for a summary, given the DUC word limit of 100 words.

2 In this case, the scores in a graph are defined to be converged if the value change over one iteration for all nodes is under the convergence threshold of 0.0001.

	ROUGE-1	ROUGE-2
TextRank (raw words)	41.64	17.17
+ stemming & stop word removal	41.46	17.40
+ postag filtering	**44.41**	**20.13**
+ all preprocessing	**44.49**	**20.25**

Table 3: The performance of default TextRank in different preprocessing configurations on the DUC-2002 dataset.

4.2 REPRODUCTION OF TEXTRANK RESULTS

Using only the information provided in the paper, we could not exactly reproduce the results of the TextRank algorithm for the DUC 2002 dataset, even though our implementation of the algorithm was exactly as described in the paper. After correspondence with the authors, it became clear that additional preprocessing was performed. Besides the necessary sentence and word tokenization steps, the authors also performed stemming, stop word removal, and most importantly, POS-tag filtering: only adjectives, nouns, and cardinal numbers were considered. Adding these preprocessing steps significantly improved TextRank's performance, as is shown in Table 3. We used all preprocessing for comparison in the survey from Chapter 3.

Although still not on par with TextRank's original performance, the remaining difference could have various causes. Firstly, we used ROUGE version 1.5.5, whereas Mihalcea and Tarau (2004) probably used version 1.4.2, as it was the newest version of the software in 2004. It is known that older versions of ROUGE are easier to fool for several reasons (Sjöbergh, 2007). Secondly, we used the POS-tagger and stemmer of Blendle's preprocessing pipeline, which might have slightly altered the final result. Finally, the original TextRank paper might have used even more hidden tuning, as the original authors indicated that they could not remember the complete procedure.

4.3 EXPERIMENT

Using our reproduction of TextRank with additional preprocessing (stemming, stop word filtering, POS tag filtering), we obtain a model that acts as a robust summarization method, both on an SDS and MDS task (as seen in Chap-

		ROUGE-1	ROUGE-2
	TextRank (raw words)	41.64	17.17
	TextRank (complete)	**44.49**	**20.25**
Google	Additive	42.40	17.98
	IDF-reweighted	**44.40**	**20.05**
	SIF	42.43	17.99
Blendle	Additive	42.82	18.19
	IDF-reweighted	**44.24**	**20.01**
	SIF	43.63	19.23

Table 4: The performance of default TextRank in different preprocessing configurations compared with word2vec-enriched TextRank on the DUC-2002 dataset, using Google's and Blendle's news vectors.

ter 3). Although this already is a nice result, it would be interesting to find a way to make TextRank less dependent on preprocessing, while preserving or even improving its performance. We hypothesize that the use of sentence embeddings should make this possible: using cosine similarities between sentence embeddings instead of normalized word overlap, we avoid additional preprocessing, while keeping a robust similarity measure for TextRank.

We will attempt to enrich TextRank with three different forms of sentence embeddings: a simple additive embedding, an IDF-reweighted additive embedding, and the SIF embedding (as discussed in Section 2.2.3). For the SIF embeddings, the first principal component and the word emission probabilities were fitted on all sentences in Blendle's English article base[3]. Each sentence embedding form is evaluated with both Blendle's own and Google's word embeddings. As with the original TextRank paper, the DUC-2002 dataset is used for evaluation. Although more sophisticated embeddings could have been used (Socher et al., 2011; Kenter et al., 2016), this set of embeddings should already give an intuition about the effectiveness of using sentence embeddings for TextRank.

The sentence selection and evaluation settings are adopted from the last study: we allow every method to slightly overshoot the 100 word limit, and

3 In other words, it was fitted on the same dataset that was used for fitting the Blendle word2vec model.

report ROUGE F-scores for evaluation, not using ROUGE's stemming or stop word removal options. The results of the experiment are shown in Table 4.

4.4 RESULTS & DISCUSSION

In Table 4, we see that additive sentence embeddings already perform better than TextRank without preprocessing, but are still outperformed by the complete version of TextRank. SIF embeddings are also outperformed by TextRank's original similarity measure. Substitution of the IDF-reweighted sentence embeddings of both the Google and the Blendle word2vec model perform on par with TextRank's original algorithm on both ROUGE-1 and ROUGE-2 scores, however. This is an interesting result, as we do not use any additional preprocessing to achieve this performance level, as opposed to the original TextRank algorithm.

Also note that, similar to the results in Chapter 3, Blendle's and Google's word2vec model perform quite similar, even though Blendle's model is trained on a significantly smaller dataset. As already mentioned in Chapter 3, this suggests that the quality of Blendle's training data is relatively high, and makes our model an interesting one, especially for news-related tasks.

Even though the TextRank method using IDF-reweighted sentence embeddings does not outperform the original one, the DUC-2002 example in Table 5 indicates that it is more capable of selecting important sentences than the original TextRank in at least some cases. In this example, the raw version of TextRank and the complete TextRank do not select the sentence at position 1, which arguably contains the main message of the article. The IDF-reweighted word2vec version of TextRank, however, succeeds in detecting the importance of this sentence. This might be explained by the fact that this sentence is semantically similar to other sentences, while using other words to convey the message. The embedding-based version of TextRank would be able to detect the similarity between sentences, while the original TextRank method would not. Note that the *LEAD* baseline also captures a lot of central information by simply selecting the beginning of the article. This again indicates that incorporating sequential information in a summarization method is beneficial to its performance.

Although the substitution of word2vec-based embeddings into TextRank improves the algorithm to some extent, the improvement is far from satisfactory. How could we use the added information of word2vec-based embeddings ef-

fectively for sentence selection? We hypothesize that only considering the cosine similarity between embeddings is too limited. A model that goes beyond this might be more effective. In the next chapter, we consider a completely different type of model, that is capable of interpreting all embedding features separately, while also considering the sequential property of textual data.

Reference summaries:

Aspirin can be a significant factor in reducing the number of heart attacks. Taking a single aspirin every other day could cut the risk of a heart attack in half. When used with a clot-dissolving drug given within 24 hours of a heart attack, the chance of a second heart attack is reduced from 13 percent to 8 percent. Many doctors still are not using clot-dissolving drugs but it is felt that, in the future, their use will be considered safe enough to allow paramedics to administer them to patients on the way to the hospital. Using a clot dissolver and aspirin could save 25,000 lives a year.

A large study has shown that taking a single aspirin every other day could cut heart attack risk in half. This was one of several reports that show doctors can prevent heart attacks as well as treat them. Doctor's also found that one aspirin and a single dose of clot dissolving given within 24 hours of heart attack dramatically cut the risk of a second fatal heart attack. Other researchers determined that some heart attack victims could be released from the hospital as soon as three days after their heart attacks. After 6 months, not one death was reported in the early discharge group.

TextRank (raw): *sentence position 3, 6, 19*

In the most recent report, doctors found that one aspirin tablet and a single dose of a clot-dissolving drug given within 24 hours can dramatically cut the risk of a second fatal heart attack. 'The real discovery is that treatment for patients is going to reduce in-hospital mortality substantially', says Richard Peto of Oxford University, one of the study's authors. 'We have to know how to respond quickly, and we have to have the courage to use these drugs in the absence of definitive diagnosis', says Dr. Burton Sobel of Washington University in St. Louis, a pioneer in the testing of the clot-dissolvers.

TextRank (complete): *sentence position 3, 4, 21, 22*

In the most recent report, doctors found that one aspirin tablet and a single dose of a clot-dissolving drug given within 24 hours can dramatically cut the risk of a second fatal heart attack. Without the treatment, patients had a 13 percent chance of dying from a second heart attack within five weeks, the researchers found. While most public attention has focused on this new ability to prevent heart attacks, researchers have also made important progress in speeding the recovery of patients who survive heart attacks. Researchers at the University of Michigan recently determined that some heart attack victims can be released from the hospital as soon as three days after their heart attacks.

TextRank (IDF-reweighted Google/Blendle word2vec): *sentence position 1, 3, 4, 21*

A study of more than 22,000 male doctors had shown that taking a single aspirin every other day could cut heart attack risk in half. In the most recent report, doctors found that one aspirin tablet and a single dose of a clot-dissolving drug given within 24 hours can dramatically cut the risk of a second fatal heart attack. Without the treatment, patients had a 13 percent chance of dying from a second heart attack within five weeks, the researchers found. While most public attention has focused on this new ability to prevent heart attacks, researchers have also made important progress in speeding the recovery of patients who survive heart attacks.

LEAD: *sentence position 0, 1, 2, 3*

Aspirin sales jumped 41 percent for a few weeks earlier this year, not because of a rash of headaches. A study of more than 22,000 male doctors had shown that taking a single aspirin every other day could cut heart attack risk in half. The study was one of a series of recent reports showing for the first time that doctors are able not only to treat heart attacks, but also to prevent them. In the most recent report, doctors found that one aspirin tablet and a single dose of a clot-dissolving drug given within 24 hours can dramatically cut the risk of a second fatal heart attack.

Table 5: System summaries for the raw version of TextRank, the complete version, and the IDF-reweighted word2vec-based TextRank method on an article from the DUC-2002 dataset. Sentence positions are reported for every system summary. Reference summaries are also shown for comparison. Note that both word2vec models result in the same summary in this case. The *LEAD* baseline from Chapter 3 is also shown. For this article, the embedding-based version of TextRank detects the importance of sentence 1, whereas the original TextRank does not.

5

STUDY 3: A NEW ALTERNATIVE: RECURRENT NEURAL NETWORKS

In this chapter, we investigate the utility of recurrent neural networks (RNNs) for extractive single document summarization. We will first discuss the challenge of outperforming the *LEAD* baseline in Section 5.1. Then we will explain how and why RNN architectures work, why they are a fit for the current problem, and the model used for our experiments in Section 5.2. We then discuss our experiments and their results in Section 5.4, and finally discuss the outcomes in Section 5.5.

5.1 THE CHALLENGE: OUTPERFORMING LEAD

In Chapter 3, we already showed the strong performance of the simple *LEAD* baseline on the DUC-2002 dataset. Allowing the *LEAD* baseline to slightly overshoot the 100 word limit like all other methods,[1] its SDS performance is superior to all other reported results (including the results from Chapter 4), as shown in Table 6.

This suggests that sentences close to the beginning of an article are more likely to be important. None of the other previously mentioned methods

	ROUGE-1	ROUGE-2
TextRank	44.49	20.25
TextRank IDF-reweighted (Google)	44.40	20.05
TextRank IDF-reweighted (Blendle)	44.24	20.01
LEAD	**46.00**	**22.34**

Table 6: ROUGE F-scores of best-performing TextRank methods and the *LEAD* baseline on the DUC-2002 dataset. *LEAD* significantly outperforms all other methods.

1 Note that many papers just use the first 3 sentences instead, which is an unfair *LEAD* baseline.

Figure 8: Normalized positional distribution of selected sentences per article. The number indicates the absolute number of sentences falling in that bin.

used the position of a sentence as a ranking feature: they neglected the sequential property of textual information. Unsurprisingly, when looking at the normalized positions of the sentences that were picked for summaries (visualized in Figure 8), we see a clear difference between the TextRank-based methods[2] and the *LEAD* baseline. Although all methods are more likely to select the first sentence, the positional distribution after the first sentence is roughly uniform for all methods except *LEAD*. In order to improve on *LEAD*, learning from its successful positional distribution, we should use a method that takes into account the sequential property of text, and is still able to exploit sentence embeddings. In fact, learning from Study 2, we would prefer a method that does not only use embeddings to compute similarities, but considers all embedding features separately. Recurrent neural network architectures exactly fit this description.

5.2 RECURRENT NEURAL NETWORKS FOR SUMMARIZATION

5.2.1 *RNN architectures: background*

In order to explain the network architecture in Section 5.2.2, some background concepts should be discussed first. In this section, we discuss the basic RNN, the long short-term memory (LSTM) network, the encoder-decoder architecture, and the use of attention mechanisms in recurrent architectures. Every concept builds on the previous concepts. Some of the figures in this section were adopted from Olah (2015), who wrote a great in-depth discussion on RNNs and LSTM networks.

2 In fact, all previously mentioned methods have a distribution similar to that of TextRank.

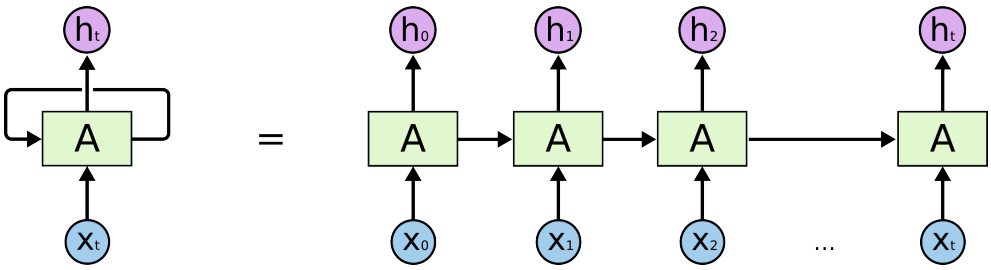

Figure 9: An 'unrolled' representation of a recurrent neural network. Note that all cells share the same weights; only the input per time step is different, and partly dependent on the previous step. Adopted from Olah (2015).

5.2.1.1 *Basic RNN*

The basic RNN structure is a simple one. Its internal structure is shown in Figure 10. Traditionally, it is a 1-layer neural network with a nonlinear activation (e.g. a *tanh* function). At each time step n of a sequence, it takes the value of that time step (say, the nth word of a sentence) as its input. The property that makes the network recurrent, is the fact that it does not only take the nth input of the sequence, but also its own output of the previous step as its input, as shown in Figure 9. This is a powerful dependency: it allows the network to 'carry' information in its hidden state about previous inputs, in order to incrementally 'learn' what the sentence means. If, for example, we would want to classify the sentiment of the sentence '*I tend not to be happy*', the network could learn to carry the information given by the word *not* until it encounters the word *happy*, classifying the whole sentence as negative.

Due to the recurrent property, backpropagation also goes backwards through all time steps. This is usually called backpropagation through time (BPTT).

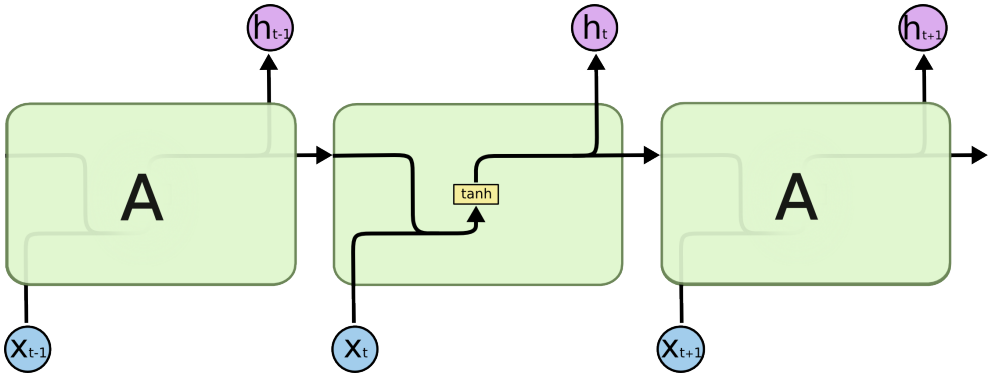

Figure 10: The internal structure of an RNN. Adopted from Olah (2015).

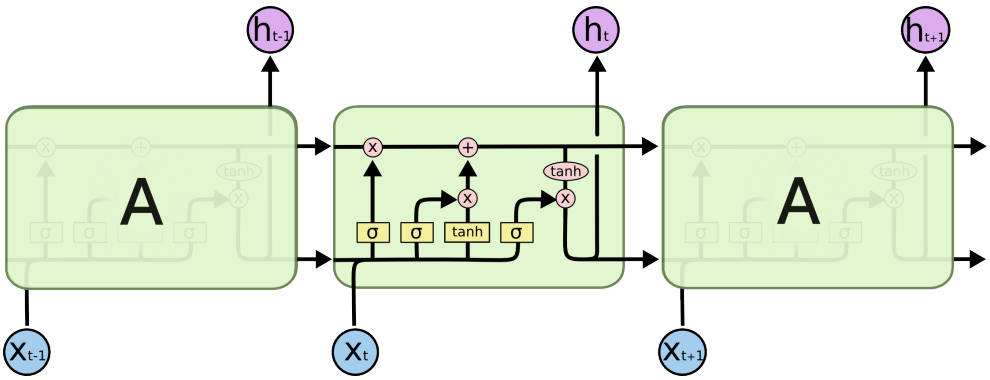

Figure 11: The internal structure of an LSTM network. Adopted from Olah (2015).

BPTT backpropagates through the whole sequence, summing the gradients of the network weights of each step. Taking sentences as inputs, it is not uncommon to have 20 time steps. Due to this 'depth' of the network, two well-known problems arise: the *vanishing gradient problem* and the *exploding gradient problem*. Although a full discussion of these problems is not in the scope of this explanation, it arises from the fact that, due to the derivative of the activation function, the time step gradients shrink or grow exponentially through the time steps. This was already discovered by Hochreiter (1991). Exponentially growing gradients are easily solvable by gradient clipping; exponentially shrinking gradients are a bigger issue. The result is that only the last few time steps have a significant effect on the weight updates, thereby losing the ability to learn long-range dependencies. The most commonly used solution for this problem is the long short-term memory (LSTM) network.[3]

5.2.1.2 *LSTM networks*

LSTM networks (Hochreiter and Schmidhuber, 1997) inherit their recurrent structure from simple RNNs, but have a far more complex internal structure. This internal structure is represented in Figure 11. Although we will not go over every operation of the internal structure, we will give a high-level overview of its function. The structure is specifically designed to overcome the problem of vanishing gradients, making it possible to learn long-term dependencies. The most important difference is the fact that an LSTM

3 Gated recurrent units (Cho et al., 2014) are a more recent successful RNN architecture, which is more efficient and performs similarly to LSTM networks. As LSTM networks still appear to be more successful for excessively long-range dependencies (i.e. > 10 time steps), we will solely focus on this architecture in this thesis.

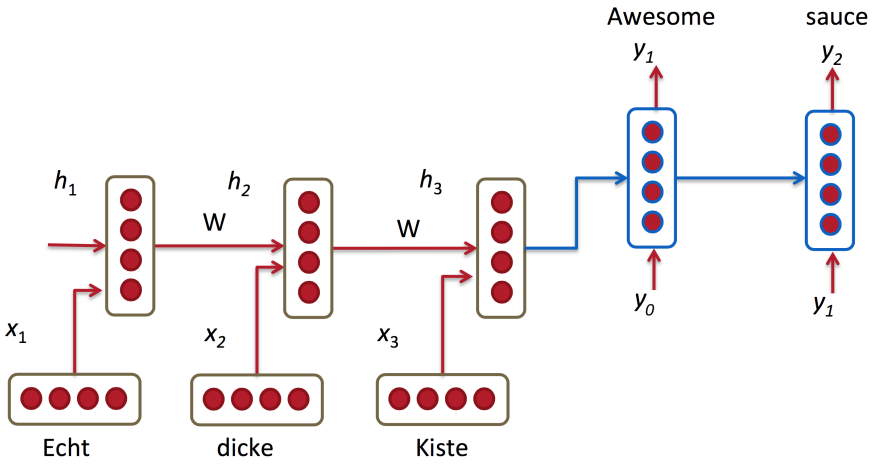

Figure 12: The encoder-decoder architecture. Adapted image from lecture.[5]

network maintains a *cell state* vector (i.e. the upper horizontal arrow running through the cell in Figure 11). This cell state is relatively stable over various time steps, and is influenced by the input and hidden state vectors through a forget ($\times$) and input ($+$) gate, which toge:ther decide what information of the cell state is updated by what information of the previous hidden state and input vectors. This updated cell state is then given as input to the next time step. It is also used together with the hidden state to decide what the 'normal' RNN output should be.[4] During training, the LSTM optimizes all these internal operations (again using BPTT) in order to learn long-range dependencies specific to the data that are useful for the task at hand.

5.2.1.3 *The encoder-decoder architecture*

Let us assume that the current task is to translate a German sentence into an English one. In this case, the task is not just to return one output based on the sequence of inputs, but an internally dependent sequence of outputs instead. We could try to simply train the output of every time step to be a word of the English sentence, but this does not work well in practice; as the word order is different for both languages, information about the full English sentence is needed before starting to translate. We would thus like

4 Note that the LSTM thus requires 3 inputs (previous cell state, previous hidden state, and input vector) and creates two outputs (the new output/hidden state and the new cell state).

5 `http://cs224d.stanford.edu/lectures/CS224d-Lecture8.pdf`

to have a model that is trained to 'encode' the German sentence in such a way that we can use its encoding to construct an English sentence with the same meaning.

This is the idea behind the encoder-decoder architecture (Cho et al., 2014; Sutskever et al., 2014), illustrated in Figure 12. It consists of two LSTM networks (with completely separate weights); one network (the *encoder*) that learns to encode the input sequence into a fixed-length vector, and one (the *decoder*) that learns to generate a sequence of outputs based on the result of the encoder. At every time step, the decoder receives its previous final output[6] and, as usual, its previous output state as input. It thus generates the most likely sequence given the decoder's weights, solely based on the fixed-length encoding.

Note that although machine translation is a common application for the encoder-decoder architecture, it is suitable for any task that has both a sequential in- and output.[7] This includes sequence labeling tasks, as we will see later in this chapter.

5.2.1.4 *Attention*

Using the encoder-decoder model, it is now possible to construct an English sentence from an encoded German sentence. As mentioned before, the decoder has very limited information for constructing a sentence; given the fixed-length encoding of the German sentence, it generates the most likely English sentence. Looking at Figure 12, this fixed-length vector is an obvious bottleneck in the information transferred from the encoder to the decoder.

Attention mechanisms (Bahdanau et al., 2014) alleviate this problem by allowing the decoder to attend to different inputs at each time step. A separate feedforward neural network assigns an attention weight to the encoder outputs of each encoder time step, given the previous decoder output. These weights are normalized, and the weighted average over the encoder outputs concatenated to the decoder's own previous output is the current decoder input. This mechanism is called *soft attention*, as the decoder is allowed to attend some encoder outputs more than others, besides its normal decoder behavior of receiving its own previous final output as input. This model is the current state-of-the-art model in the field of machine translation.

6 Note that the final output might be different from the decoder's 'raw' output; usually, the final output is a softmax over the time step's raw output vector.

7 This is why the equivalent term for encoder-decoder model is *sequence-to-sequence* model.

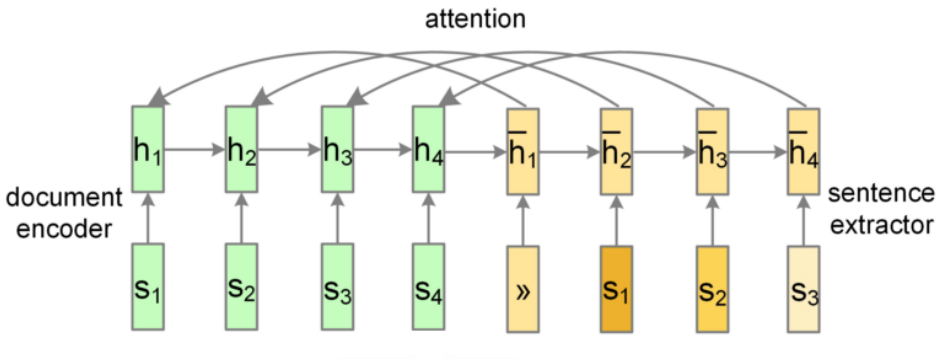

Figure 13: The extractive summarization architecture proposed by Cheng and Lapata (2016). Image is an adapted version from the original paper.

5.2.1.5 *RNNs for extractive summarization*

Cheng and Lapata (2016) proposed an extractive summarization method[8] based on the encoder-decoder model with attention as proposed by Bahdanau et al. (2014). Their model is visualized in Figure 13. As opposed to the machine translation example, this means that our RNN encoder is not a sentence encoder reading words, but a document encoder reading sentences. In order to obtain sentence embeddings from word embeddings, Cheng and Lapata (2016) use a convolutional sentence encoder, which we will not address; we use another, more simple and generic sentence representation in our own model, which is discussed in Section 5.2.2. Given the sequence of sentences, the task is now a sequence labeling task: for every sentence in the sequence, the decoder predicts whether sentence should be included in the summary. As shown in Figure 13, the label of the nth sentence s_n is predicted by a feedforward neural network $f(\cdot)$, taking the concatenation of the current decoding output $\bar{h}_n$ and the corresponding encoding output h_n as the input[9]:

$$p_n(y_n = 1) = f(\bar{\boldsymbol{h}}_n : \boldsymbol{h}_n).$$

This concatenation could be seen as a *hard attention mechanism*, fully attending the corresponding encoding state instead of softly attending some

8 They actually proposed both an extractive and an abstractive method, but we will only consider the extractive method; abstractive summarization is not the subject of this thesis.

9 Note that we use ':' as the concatenation operator.

encoding states. The next decoding state $\overline{\boldsymbol{h}}_{n+1}$ is then calculated as follows[10]:

$$\overline{\boldsymbol{h}}_{n+1} = \text{decoder}(p_n \boldsymbol{s}_n,\ \overline{\boldsymbol{h}}_n).$$

This means that the decoder has access to the raw input sentence embedding of the previous time step, multiplied by the estimated 'pick probability' of that sentence.

The dataset used for training the model will later be described in Section 5.3, as we used the same dataset for training our model.

5.2.2 *Our architecture*

Although the RNN model proposed by Cheng and Lapata (2016) is not easily reproducible[11], we propose a new model based on their model architecture. The transformations to their model to get to our model architecture are discussed below.

Firstly, the convolutional layer used in their model is only necessary to provide word-level attention for abstractive summarization. As our method is aimed at extractive summarization only, we use static sentence embeddings instead of a convolutional network to construct sentence embeddings from word embeddings; the input sentence embedding is simply the average of all word embeddings in the sentence. Although this is a restriction of the model, we expect that such a static sentence embedding already provides all information for the RNN architecture to predict the sentence labels, while being much simpler and faster to implement. It also simplifies our model by isolating the RNN architecture as the only 'learning' component of the model.

Secondly, we slightly altered the fixed attention mechanism. Instead of multiplying the raw sentence embedding of the previous time step with the pick probability of that sentence, we choose to concatenate the pick probability with the sentence embedding. The decoder output is now calculated as follows:

$$\overline{\boldsymbol{h}}_{n+1} = \text{decoder}(p_n : \boldsymbol{s}_n,\ \overline{\boldsymbol{h}}_n).$$

10 Note that for simplicity, the LSTM cell state is omitted from both sides of this equation.

11 Although the authors published their code on GitHub, the procedure in that code did not match the paper. The authors did not respond to questions about this via email.

This is more similar to the original attentional approach by Bahdanau et al. (2014): the pick prediction is the final output of the decoder at that time step, which should thus be given as input to the next decoding time step. Besides, this provides the decoder with more information: instead of multiplying the pick probability, we provide it explicitly, thereby giving the decoder the opportunity to learn the relation between these terms itself. Note that we still have a look-back into the corresponding encoding output, directly through the attentional feedforward neural network, exactly like Cheng and Lapata. Besides the aforementioned benefit of look-back, this fixed attention mechanism also provides a powerful backpropagation path directly from the label to the corresponding encoding state, thus bypassing the standard BPTT path. This allows the model to also adjust its encoder weights based on the output provided by the corresponding encoder state, besides the 'normal' path through the final hidden state of the encoder.

For more details on layer sizes, hyperparameter settings, and prevention of overfitting, see Section 5.4.1. Compared to the aforementioned methods in this thesis, this is the first type of model that is able to utilize the sequential property of text, and separately interpret the 300 features of the embedding space. This gives our model the possibility of only paying attention to some combination of input features, depending on the sentence embeddings it has seen up to that embedding.

5.3 DATASET

Since we train our RNN architecture in a supervised manner, we need a labeled dataset that is large enough to optimize the large number of parameters in our model. We use the DailyMail dataset by Cheng and Lapata (2016), which contains 216483 articles[12]. All articles and their corresponding bullet point summaries provided by DailyMail were retrieved from the DailyMail website. Each article is already split into sentences and tokenized, and every sentence is labeled as 0 (should not be extracted), 1 (should be extracted), or 2 (might be extracted). For our task, all 2-labels are interpreted as 0-labels, essentially making the task a single-label sequence labeling task, predicting for every sentence whether the sentence should be included in the summary or not. On average, about 1/3 of all sentences has a 1-label. As expected,

12 Split into 193986 training, 12147 validation, and 10350 test articles.

Figure 14: Normalized positional distribution of 1-labels in a random sample of 10000 articles from the DailyMail dataset. The number indicates the the probability of a sentence to occur in that position (of 20 position bins).

the positional distribution of 1-labeled sentences is skewed towards the beginning of articles, as shown in Figure 14. We expect our RNN model to learn a similar positional distribution.

It is important to note that not all articles were labeled manually; Cheng and Lapata (2016) designed and fitted a rule-based system to automatically label all sentences based on the unigram and bigram overlap between the article sentences and the Dailymail summaries, the sequential position of the sentences in the document, and the number of entities in the sentence. The weights of the rules were fitted on a subset of 9000 documents that were labeled manually. On a left-out test set of 216 labeled documents, the rule-based sentence labeler reached an accuracy of 85%.[13] Despite the fact that not all labels in the dataset were assigned manually, we will still refer to them as the true labels later in this chapter.

For evaluation, we use ROUGE F-scores on the DUC-2002 dataset which we also used in our previous studies.

5.4 EXPERIMENT & RESULTS

5.4.1 *General settings (hyperparameters, layer sizes, etc.)*

In this section, we mention some general settings that are the same for all model versions discussed in Sections 5.4.2–5.4.3.

All versions of the RNN model were trained on an NVidia K80 GPU on the Google Cloud Platform. To build the model, we used PyTorch[14], a relatively

13 Note that this means that the upper bound of accuracy of any sentence extractor on the full set is also expected to be around 85%.

14 `pytorch.org`

Layer	Size
Input (= embedding)	300
Encoder hidden state	500
Decoder hidden state	500
Encoder cell state	500
Decoder cell state	500
Attention hidden layer	500

Table 7: An overview of layer sizes in the RNN model.

new, actively developed, open-source deep learning framework in Python written by Facebook. This framework provides more readable syntax than TensorFlow, especially for variable-length sequential data. It is also heavily optimized for use on GPUs. The code for our model is available on GitHub[15].

We tested all RNN model versions with both the Blendle and the Google word2vec model as its basis. Training was stopped when the loss decrease on the validation set was less than .01, taking the last epoch with a larger loss decrease as the final model. This means that unless stated otherwise, all models were run for 1 epoch; in almost all cases, training for longer than 1 epoch only led to overfitting. As our model uses the predicted previous label as input for its next label prediction, the error in prediction may incrementally increase, especially at the start of training. To control for this, we used teacher forcing: with a probability of 0.5, the real label was given as input to the next time step, instead of the predicted label. While the cell state and hidden state of the LSTM encoder at the beginning of every sequence (i.e. document) are initialized with zero matrices the first time, these initial states are also trained. The attention network was instantiated as a feedforward neural network with one hidden layer followed by a rectified linear unit (ReLU) activation layer (Nair and Hinton, 2010). All models were trained stochastically (i.e. with a batch size of 1) with Adam (Kingma and Ba, 2014), setting the initial learning rate to 10^{-3}. To prevent overfitting, the input dropout probability (Srivastava et al., 2014) was set to 0.3. Layer sizes are summarized in Table 7.

15 https://github.com/blendle/research-summarization

	ROUGE-1	ROUGE-2
TextRank	44.49	20.25
TextRank IDF-reweighted (Google)	44.40	20.05
TextRank IDF-reweighted (Blendle)	44.24	20.01
LEAD	46.00	22.34
RNN (normal training; Google)	**46.56**	**22.75**
RNN (normal training; Blendle)	**46.85**	**23.01**

Table 8: ROUGE F-scores of the first RNN-based model and the strongest SDS baselines on the DUC-2002 dataset.

5.4.2 *Run 1: Normal training*

Using the model with the settings discussed in Section 5.4.1, we trained the model for 1 epoch[16] with both the Blendle and the Google word2vec model. The relation between training loss and training accuracy and the loss over training iterations is shown in Figure 15. The performance evaluation against the best TextRank-based models and the *LEAD* baseline is shown in Table 8. The Blendle RNN model achieves the best performance on both the ROUGE-1 and ROUGE-2 scores. Its ROUGE-1 score is significantly better than the *LEAD* baseline.

Based on Figure 16, we conclude that both the Google- and Blendle-word2vec-based RNN model learned the positional distribution of the training data shown in Figure 8, and thus learns from the sequential property of

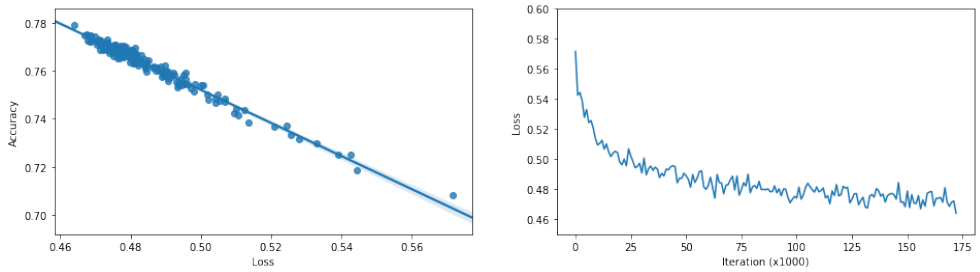

Figure 15: Plot of the relation between mean and loss (left) and the loss over training iterations (right) for the first, normal training run with the Blendle word2vec model. Run on the Google model is very comparable. Data points are means over every 1000 iterations.

16 After 1 epoch, the model already appeared to start overfitting.

Figure 16: Normalized positional distribution of selected sentences per article for the normally trained RNN models, with the *LEAD* baseline and TextRank for reference. The number indicates the absolute number of sentences falling in that bin.

the text data. This also means that the positional distribution is heavily skewed towards the beginning of article. We might interpret our model as an alternative for *LEAD*, that learns what sentences to 'skip' based on the predicted labels and the other sentences in the document. This indeed seems to be an accurate description of what the RNN models do. Table 9 shows example summaries for TextRank, the *LEAD* baseline, and both RNN models. As seen in many articles, the first sentence of the example article contains important information. TextRank fails to select this sentence, whereas both *LEAD* and the RNN rankers include it in the summary. Both RNNs, however, seem to exclude unimportant sentences at the beginning of the example article, thereby saving space for more important sentences that appear later in the article.[17] This might be a (partial) explanation for the performance improvement over the *LEAD* baseline.

As both the RNN rankers and TextRank-based methods achieve good scores on DUC-2002, we hypothesize that both these methods select a different, but interesting set of sentences. In order to make our RNN ranker deviate from *LEAD* even further, we now try to combine the TextRank result and the *LEAD* result into one ranker, without compromising the performance.

17 Note that the selected sentences by the RNN rankers later on in the example article are not always informative; the Blendle word2vec-based ranker seems to do a better job than the Google word2vec-based ranker in this case.

Reference summaries:

In Kuwait, with the sound of gunfire, artillery and military jets streaming overhead, the entire foreign community is staying indoors. The crisis began on July 17 when Saddam accused the Kuwaitis of trying to wreck his debt-ridden economy with oil overproduction and of stealing Iraqi oil from a border oil field. He also demanded that Kuwait turn over the Gulf island of Bubiyan and forgive multi-billion dollar loans incurred by Iraq during its 1980-1988 war with Iran. The invasion came a day after talks between Iraq and Kuwait collapsed. Today's fighting sent oil prices surging in international markets.

Iraqi troops, led by the elite Republican Guard invaded Kuwait early today as Iraq claimed it was responding to the request from 'the interim government of free Kuwait' to defend the revolution and the Kuwaiti people. Kuwaiti forces failed in their attempt to repel the invaders with heavy artillery. The rattle of automatic weapons and explosions in rapid succession rudely awakened residents. The British Embassy was hit by shellfire and the Sheraton and Hilton hotels evacuated their residents. Prime Minister Sheik Saad al-Abdullah al-Sabah appealed on Kuwaiti radio Arab support. Today's fighting quickly sent oil prices surging in international markets.

TextRank (complete): *sentence position 16, 21, 25, 26, 30*

Iraq has an estimated 1 million troops — including reservists — among its 16 million people, compared to 20,300 troops in Kuwait. Saddam had threatened military action against Kuwait and the United Arab Emirates for exceeding their cartel production quotas and driving crude prices down. The crisis began July 17 when Saddam accused Kuwait of trying to wreck the debt-ridden Iraqi economy with oil overproduction and began massing troops on Kuwait's border. He also accused Kuwait of stealing $2.4 billion in oil drilled from the border oil field, Rumailah. Iraq borrowed $10 billion to $20 billion from Kuwait wants that debt forgiven.

LEAD: *sentence position 0, 1, 2, 3, 4, 5, 6, 7, 8*

The British Embassy was hit by shellfire and the Sheraton and Hilton hotels evacuated their clientele, sources said. Swiss charge d'affaires Franco Bessoni said on French radio that the palace was taken after bombardment by Iraqi MiG jet fighters. 'You can see black smoke over there . . . No one dares to go out. You can hear gunfire. The entire foreign community is staying indoors', he said. 'It's chaos, military jets are flying over all the time', Kathy McGregor, a Canadian, said by telephone at about midday. She said she could still hear artillery fire. The air space over Kuwait was closed, with airliners turned away.

Google word2vec-based RNN: *sentence position 0, 1, 6, 19, 20*

The British Embassy was hit by shellfire and the Sheraton and Hilton hotels evacuated their clientele, sources said. Swiss charge d'affaires Franco Bessoni said on French radio that the palace was taken after bombardment by Iraqi MiG jet fighters. 'It's chaos, military jets are flying over all the time', Kathy McGregor, a Canadian, said by telephone at about midday. The price of North Sea Brent crude for September delivery was quoted early today at $22.30 a barrel against Wednesday's London close of $20.40. Iraq, the world's second-largest oil producer, had demanded higher oil prices at last month's OPEC oil cartel meeting in Geneva.

Blendle word2vec-based RNN: *sentence position 0, 1, 6, 11, 14*

The British Embassy was hit by shellfire and the Sheraton and Hilton hotels evacuated their clientele, sources said. Swiss charge d'affaires Franco Bessoni said on French radio that the palace was taken after bombardment by Iraqi MiG jet fighters. 'It's chaos, military jets are flying over all the time', Kathy McGregor, a Canadian, said by telephone at about midday. It said its troops — no numbers were given — were in Kuwait to defend the revolution and the Kuwaiti people. Kuwaiti forces had engaged the Iraqi invaders with heavy artillery in their failed attempt to repulse them, Kuwaiti officials said earlier.

Table 9: System summaries for the complete version of TextRank, the first RNN ranker (both the Google en Blendle word2vec version), and the *LEAD* baseline on an article from the DUC-2002 dataset. Reference summaries are also shown for comparison. The RNN rankers seem to learn which sentences to skip when reading the article.

5.4.3 *Run 2–6: Co-/pre-training with TextRank labels*

In an attempt to combine the strengths of both TextRank and our first RNN ranker, we combine the training of an RNN model on the true DailyMail labels with training on unsupervised TextRank labels. This allows for semi-supervised[18] training of an RNN ranker, thereby deviating from the automatically assigned DailyMail labels as the only source for training. This semi-supervised approach is a new contribution to the field of extractive summarization, giving endless possibilities of combining the behavior of several powerful summarization systems into one recurrent neural network. In this thesis, we only use TextRank for unsupervised label production, but note that any ranker (or combination of rankers) could be used for this unsupervised signal.

We performed semi-supervised training of the RNN model in different ways. We discriminate between pre-training and co-training: pre-training denotes training on TextRank labels only before training on true labels, while co-training denotes using either TextRank or true labels randomly per article, given a predefined ratio. The TextRank labels were constructed by labeling the highest ranked $1/3$ of the sentences with a 1-label. We evaluate the performance[19] of all different semi-supervised models on the DUC-2002 dataset.

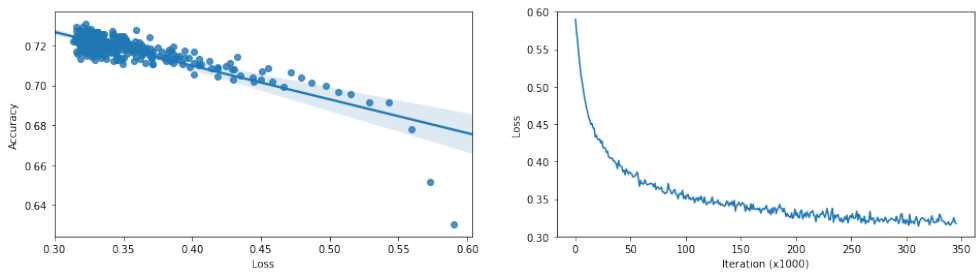

Figure 17: Plot of the relation between mean and loss (left) and the loss over training iterations (right) on the TextRank pre-training with the Blendle model. Google pre-training is very comparable. Data points are means over every 1000 iterations.

18 Even though the RNN is only trained with labeled data, we use the term semi-supervised because we use an unsupervised method to construct 'mock' labels.

19 Again on both the Blendle and the Google model.

		ROUGE-1	ROUGE-2
	LEAD	46.00	**22.34**
Blendle	Normal training only	**46.85**	**23.01**
	Co-training only (0.5)	46.06	22.18
	Pre-training only	43.46	19.01
	+ normal	**46.79**	**22.97**
	+ co-training (0.5)	44.22	20.07
	+ co-training (0.25)	**46.27**	**22.57**
Google	Normal training only	**46.56**	**22.75**
	Co-training only (0.5)	45.56	21.46
	Pre-training only	44.15	19.77
	+ normal	**46.39**	**22.70**
	+ co-training (0.5)	45.11	20.90
	+ co-training (0.25)	45.73	21.79

Table 10: ROUGE F-scores on all different RNN model versions and the *LEAD* baseline.

These models include:

- Pre-training only
- Pre-training, then 1 normal epoch
- Pre-training, then co-training (TextRank label probability: 0.5)
- Pre-training, then co-training (TextRank label probability: 0.25)
- Co-training only (TextRank label probability: 0.5)

Pre-training converged after 2 epochs on the Blendle model and 3 epochs on the Google model. The relation between pre-training loss and pre-training accuracy and the loss over pre-training iterations is shown in Figure 17[20]. Aside from pre-training, all training converged after 1 epoch. The evaluation results on DUC-2002 are shown in Table 10; a visualization of all ranker distributions is given in Figure 18.

Let us first consider the ranking results directly after pre-training on the TextRank labels (i.e. the 'pre-training only' rankers). In Figure 18, we

20 We only visualize the loss during pre-training, as the loss and accuracy on co-training are a random mix on TextRank and normal labels.

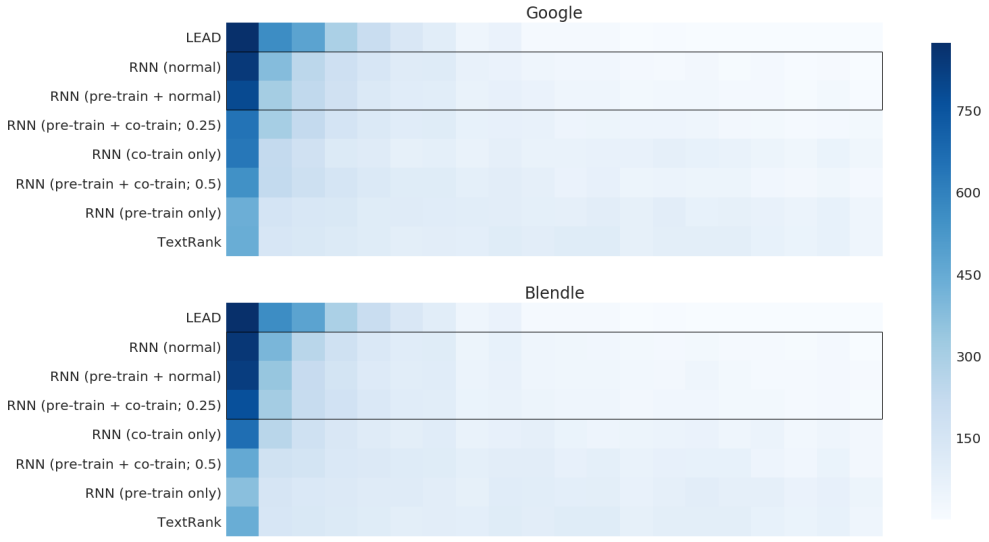

Figure 18: Normalized positional distribution of selected sentences per article for the all RNN models (both Google and Blendle), with the *LEAD* baseline and TextRank for reference. The number indicates the absolute number of sentences falling in that bin. Marked distributions achieve the best performance on DUC-2002 (see Table 10).

see that both the Blendle- and Google-based pre-trained RNN learn a positional distribution on the DUC-2002 data comparable to that of the actual TextRank ranker. The performance on ROUGE-1 and ROUGE-2 of the Google-word2vec-based RNN even is on par with the TextRank result. We conclude that the RNN is sufficiently capable of learning the TextRank behavior.

Considering the *co-training only* result, we indeed see that this RNN learned a positional distribution that is between the normally trained RNN ranker and the *pre-training only* ranker. This is exactly what we expect, as it is trained on a combination of both labels. The *co-training only* ranker performs significantly worse than the best ranker, however. The only successful[21] model using co-training is an RNN using the Blendle word2vec model that was first pre-trained on TextRank labels, and co-trained for 1 epoch afterwards, with a TextRank label probability of just 0.25.

When focusing on ROUGE-1 scores[22] in Table 10, we see that the Blendle ranker from our first run still obtains the highest score. It is now matched

21 With successful, we mean that it performs on par with the best model.

22 We focus on ROUGE-1 scores since these are known to correlate most strongly with human summary evaluation, as we mentioned in Section 2.4.

by several pre- and co-trained rankers, which are slightly less skewed towards the beginning of articles than the normally trained rankers. It becomes clear, however, that a strongly skewed distribution is beneficial; the best rankers of the set are the rankers of which the positional distribution is most similar to *LEAD*.

5.5 DISCUSSION

We conclude that RNN rankers are most successful in using information from sentence embeddings for extractive summarization, and the only type of ranker considered in this thesis that is capable of beating the simple but strong *LEAD* baseline. They do so by combining a sequential interpretation of the data with a per-feature interpretation of the embeddings. Although the ranker purely based on the true labels achieved the best performance, a ranker that deviates stronger from *LEAD* might be preferable: we want a ranker that finds the most relevant sentences throughout the whole article, even though some bias towards the beginning seems to be justifiable.

We further conclude that the Blendle word2vec model again proves to be a small, but effective model for representing words and sentences, making it an interesting alternative for the Google News vectors in the news domain. As we used a static sentence embedding, we can also conclude that the RNN architecture is able to extract relevant information from the static sentence vectors, without using a jointly trained sentence encoder. As opposed to the architecture by Cheng and Lapata (2016), which also contains a convolutional sentence encoder, the RNN architecture is the only 'learning' component of our model.

Finally, we attempted to combine training on true labels with training on labels constructed by an unsupervised SDS technique. To our knowledge, this is the first time that unsupervised and supervised summarization techniques were combined this way. Although pre-training on TextRank proves that RNNs indeed learn from unsupervised labels, the possibilities of this semi-supervised framework are endless.

6

DISCUSSION AND CONCLUSION

The objective of this thesis was to design, implement, and evaluate an embedding-based algorithm for extractive summarization, in order to automatically find the most important set of sentences in any article. We aimed at an algorithm using word2vec-based sentence embeddings for representing sentences, as word2vec (Mikolov et al., 2013b) carries unsupervised semantic word information into the model. In order to achieve this, we first compared current state-of-the-art embedding-based extractive summarization method against each other and strong non-embedding-based baseline methods. Then we tried enriching a robust SDS algorithm from this first study, TextRank (Mihalcea and Tarau, 2004), with various forms of sentence embeddings. Although this had the positive effect of eliminating the need for additional preprocessing steps used by the original algorithm without losing performance, it did not outperform TextRank. Finally, we used a sequential model that is able to flexibly interpret sentence embeddings, inspired by the strong *LEAD* baseline and the RNN-based model by Cheng and Lapata (2016). This model outperformed *LEAD*, but also appears to be capable of approximating the behavior of other summarization methods. By introducing pre- and co-training on unsupervised summarization methods (TextRank, in our case), we provided a framework for semi-supervised extractive summarization, capable of learning 'ensembles' of methods. Our final proposed method, however, is the normally trained RNN model using the Blendle word2vec model for its sentence embeddings, as it achieves the highest performance of all models evaluated in this thesis. Although this RNN and its underlying word2vec model are only trained for summarization of English articles, training the model for Dutch articles is feasible given Blendle's data. This feasibility is discussed in Appendix A. Throughout this thesis, the word2vec model trained on Blendle's data proved to be as effective as the much larger Google News word2vec model, arguably due to the quality of the text sources. It could be an interesting alternative for the Google News model, especially in the news domain.

6.1 DISCUSSION OF RESEARCH QUESTIONS

We stated three main research questions in Chapter 1. We will summarize the answers to these questions below.

RQ1: How do the current embedding-based extractive methods compare to each other and commonly used non-embedding-based baselines?
While in theory, embeddings only add information to the model for free by unsupervisedly learning word meaning on large word corpora, it seems difficult to use this information to improve summary quality. Previously proposed embedding-based methods tend to perform worse than strong, non-embedding-based (i.e. based on TF-IDF or even word overlap) baselines on both MDS and SDS datasets. Especially TextRank (Mihalcea and Tarau, 2004), simply using word overlap, performs well on single document summarization, which is the main goal of this thesis. All methods, however, are outperformed by the simple, but very effective *LEAD* baseline on the SDS task.

RQ2: Is it possible to improve the performance of existing non-embedding-based methods by substituting embeddings?
Using the well-performing TextRank from the first question, we tried to incorporate various forms of word2vec-based sentence embeddings. We did so by using the cosine similarity between the sentence embeddings as the edge weights in the graph, instead of the originally used normalized word overlap. IDF-reweighted sums of word embeddings in the sentence performed on par with the original TextRank algorithm, without applying the additional preprocessing steps used in the original paper (i.e. POS tag filtering, stemming, stop word filtering). Although this does not improve the performance of the original method, the lack of additional preprocessing could be seen as an enhancement.

RQ3: Can a sequential model using unsupervised word embeddings improve on the strong beginning-of-article heuristic?
We proposed a recurrent neural network (RNN) model with an encoder-decoder architecture and fixed attention, inspired by Cheng and Lapata (2016). This model makes use of the sequential property of text, just like the

beginning-of-article heuristic (also known as the *LEAD* baseline), while also interpreting the sentence embeddings more flexibly than all previous models. This model outperforms the *LEAD* baseline, but still shows a positional distribution similar to that of *LEAD*. In order weaken this bias towards the beginning of articles, we propose to pre-, or even co-train our RNN model with other, unsupervised labels, such as TextRank. Although this pre-trained model does not outperform our first RNN model, it does perform on par with it, while being less biased towards the beginning of articles.

6.2 FUTURE DIRECTIONS

We would like to point out several interesting directions for further research.

First and foremost, an obvious direction is to further investigate the final semi-supervised RNN-based framework. It would be interesting to study the capabilities of the RNN model to approximate other summarization algorithms, but also to find a set of complementing summarization methods that could be used as an ensemble approximated by the RNN.

A second direction would be to try various RNN-based architectures, as there is still a lot of room for exploration.[1] A thorough evaluation of the effect of a bidirectional encoder/decoder and the use of soft instead of hard attention would be good places to start. It would also be interesting to evaluate the performance of a much simpler LSTM model, not incorporating the encoder-decoder architecture.

1 Due to time constraints, there was no more tuning of model architecture and hyperparameters involved than described in this paper.

BIBLIOGRAPHY

Armstrong, T. G., Moffat, A., Webber, W., and Zobel, J. (2009). Improvements that don't add up: ad-hoc retrieval results since 1998. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 601–610. ACM.

Arora, S., Liang, Y., and Ma, T. (2017). A simple but tough-to-beat baseline for sentence embeddings. In *ICLR*.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. In *Proceedings of International Conference on Learning Representations*.

Banerjee, S. and Pedersen, T. (2003). Extended gloss overlaps as a measure of semantic relatedness. In *IJCAI*, volume 3, pages 805–810.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Brychcın, T. and Svoboda, L. (2016). Uwb at semeval-2016 task 1: Semantic textual similarity using lexical, syntactic, and semantic information. In *Proceedings of SemEval*, pages 588–594.

Carbonell, J. and Goldstein, J. (1998). The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 335–336. ACM.

Cheng, J. and Lapata, M. (2016). Neural summarization by extracting sentences and words. *arXiv preprint arXiv:1603.07252*.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Erkan, G. and Radev, D. R. (2004). Lexrank: Graph-based lexical centrality as salience in text summarization. *Journal of Artificial Intelligence Research*, 22:457–479.

Ganesan, K., Zhai, C., and Han, J. (2010). Opinosis: a graph-based approach to abstractive summarization of highly redundant opinions. In *Proceedings of the 23rd international conference on computational linguistics*, pages 340–348. Association for Computational Linguistics.

Harris, Z. S. (1954). Distributional structure. *Word*, 10(2-3):146–162.

Hill, F., Cho, K., and Korhonen, A. (2016). Learning distributed representations of sentences from unlabelled data. In *Proceedings of NAACL-HLT*, pages 1367–1377.

Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Holm, S. (1979). A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics*, pages 65–70.

Joulin, A., Grave, E., and Mikolov, P. B. T. (2017). Bag of tricks for efficient text classification. *EACL 2017*, page 427.

Kågebäck, M., Mogren, O., Tahmasebi, N., and Dubhashi, D. (2014). Extractive summarization using continuous vector space models. In *Proceedings of the 2nd Workshop on Continuous Vector Space Models and their Compositionality (CVSC)@ EACL*, pages 31–39. Citeseer.

Kenter, T., Borisov, A., and de Rijke, M. (2016). Siamese cbow: Optimizing word embeddings for sentence representations. *arXiv preprint arXiv:1606.04640*.

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kiros, R., Zhu, Y., Salakhutdinov, R. R., Zemel, R., Urtasun, R., Torralba, A., and Fidler, S. (2015). Skip-thought vectors. In *Advances in neural information processing systems*, pages 3294–3302.

Kobayashi, H., Yatsuka, M., and Taichi, N. (2015). Summarization Based on Embedding Distributions. *EMNLP*, pages 1984–1989.

Le, Q. V. and Mikolov, T. (2014). Distributed representations of sentences and documents. In *ICML*, volume 14, pages 1188–1196.

Lin, C.-Y. (2004). Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out: Proceedings of the ACL-04 workshop*, volume 8. Barcelona, Spain.

Lin, H. and Bilmes, J. (2010). Multi-document summarization via budgeted maximization of submodular functions. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 912–920. Association for Computational Linguistics.

Lin, H. and Bilmes, J. (2011). A class of submodular functions for document summarization. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, volume 1, pages 510–520. Association for Computational Linguistics.

McCormick, C. (2016). Word2vec tutorial – the skip-gram model.

Melis, G., Dyer, C., and Blunsom, P. (2017). On the state of the art of evaluation in neural language models. *arXiv preprint arXiv:1707.05589*.

Metzler, D., Bernstein, Y., Croft, W. B., Moffat, A., and Zobel, J. (2005). Similarity measures for tracking information flow. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 517–524. ACM.

Mihalcea, R. and Tarau, P. (2004). TextRank: Bringing order into texts. *EMNLP*, 85:404–411.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Mitchell, J. and Lapata, M. (2010). Composition in distributional models of semantics. *Cognitive science*, 34(8):1388–1429.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

Nichols, T. E. and Holmes, A. P. (2002). Nonparametric permutation tests for functional neuroimaging: a primer with examples. *Human brain mapping*, 15(1):1–25.

Olah, C. (2015). Understanding lstm networks. *GITHUB blog, posted on August*, 27:2015.

Page, L., Brin, S., Motwani, R., and Winograd, T. (1999). The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab.

Pagliardini, M., Gupta, P., and Jaggi, M. (2017). Unsupervised learning of sentence embeddings using compositional n-gram features. *arXiv preprint arXiv:1703.02507*.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543.

Radev, D. R., Jing, H., Styś, M., and Tam, D. (2004). Centroid-based summarization of multiple documents. *Information Processing and Management*, 40(6):919–938.

Sanderson, M. and Zobel, J. (2005). Information retrieval system evaluation: effort, sensitivity, and reliability. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 162–169. ACM.

Sjöbergh, J. (2007). Older versions of the rougeeval summarization evaluation system were easier to fool. *Information Processing & Management*, 43(6):1500–1505.

Smucker, M. D., Allan, J., and Carterette, B. (2007). A comparison of statistical significance tests for information retrieval evaluation. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 623–632. ACM.

Socher, R., Huang, E. H., Pennington, J., Ng, A. Y., and Manning, C. D. (2011). Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In *NIPS*, volume 24, pages 801–809.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958.

Sultan, M. A., Bethard, S., and Sumner, T. (2015). Dls@ cu: Sentence similarity from word alignment and semantic vector composition. In *Proceedings of the 9th International Workshop on Semantic Evaluation*, pages 148–153.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics bulletin*, 1(6):80–83.

Yogatama, D., Liu, F., and Smith, N. A. (2015). Extractive summarization by maximizing semantic volume. In *EMNLP*, pages 1961–1966.

A

FEASIBILITY OF RNN-BASED SUMMARIZATION ON DUTCH BLENDLE ARTICLES

Although the RNN ranker is the best-performing SDS method found in this thesis, it is only trained for English. As most of Blendle's articles are Dutch, a Dutch RNN ranker would be desirable. In this chapter, we will briefly argue why training a Dutch RNN ranker is theoretically feasible given Blendle's own data.

In order to show the feasibility of training a Dutch RNN ranker, we need to show that both the RNN ranker and a well-performing, unsupervised word2vec model can be trained using Blendle's data. The construction of a dataset for training the RNN is discussed in Section A.1 and the Dutch Blendle word2vec model is addressed in Section A.2.

A.1 DATASET CONSTRUCTION

As mentioned in Chapter 5, the dataset constructed by Cheng and Lapata (2016) contains ∼200k DailyMail articles, where a small set is manually labeled, labeling most of the dataset automatically using the summaries provided by DailyMail. As for Blendle, the database contains ∼2.1M Dutch articles, ∼57.5M sentences, or ∼708.3M words, probably making it the largest high-quality[1] text corpus for Dutch. Every article text contains metadata about the type of content, dividing an article into a title, intro, paragraphs etc. Article intros usually provide a rough summary of the article, making them comparable to the DailyMail summaries. Some examples of Dutch article intros are shown in Table 11. About 0.4M articles (≈17.8% of all articles) contain an intro that is longer than 5 words.[2] This arguably makes the set of article-intro pairs from the Blendle data a Dutch equivalent to the English dataset of Cheng and Lapata (2016).

1 Not considering Twitter corpora, for example

2 This is a heuristic cut-off to exclude non-summary intros.

Voor het eerst in zes jaar wijzen alle signalen bij het Amerikaanse beurshuis Morgan Stanley erop dat het tijd is om Europese aandelen te kopen.
Autojournalist Fleur Baxmeier geeft gas in nieuwe auto's. Deze week: Porsche 911 Targa.
Tieners kunnen vaak nog zonder problemen zelfstandig alcohol of sigaretten kopen.
Na een keurmerk voor eieren en vlees moet er ook een Beter Leven Keurmerk komen voor zuivel.
Twee maanden na de opening is er nog altijd geen actie op het Leeuwarder stadsstrand. Heb toch een beetje geduld met ons, vragen de eigenaren.
De eredivisie is dit seizoen ongemeen spannend. PSV en Ajax spelen 20 maart tegen elkaar. De titelkandidaten beoordeeld op acht factoren. Conclusie: PSV is in het voordeel.

Table 11: Some ramdomly sampled article intros.

A.2 WORD2VEC MODEL

The ∼57.5M high-quality Dutch sentences in the Blendle database are a unique source for training a Dutch word2vec model. We trained a word2vec skip-gram model using the same settings as the English word2vec model used throughout this thesis. Given the fact that the English Blendle word2vec model performed on par with Google's word2vec model, we hypothesize that the Dutch word2vec model should also perform well as foundation for the RNN ranking model. It might even be the best-performing Dutch word2vec model due its uniquely large and clean training source. This model will be publicly available on GitHub[3] in order to facilitate evaluating this claim.

As it seems possible to construct both a summarization dataset and a word2vec model comparable to that of Cheng and Lapata (2016), we conclude that it is feasible to train a well-performing RNN summarizer for Blendle's Dutch articles.

3 https://github.com/blendle/research-summarization