Extractive Summarization using Continuous Vector Space Models

Mikael Kågebäck, Olof Mogren, Nina Tahmasebi, Devdatt Dubhashi
Computer Science & Engineering
Chalmers University of Technology
SE-412 96, Göteborg
{kageback, mogren, ninat, dubhashi}@domain

Abstract

Automatic summarization can help users extract the most important pieces of information from the vast amount of text digitized into electronic form everyday. Central to automatic summarization is the notion of similarity between sentences in text. In this paper we propose the use of continuous vector representations for semantically aware representations of sentences as a basis for measuring similarity. We evaluate different compositions for sentence representation on a standard dataset using the ROUGE evaluation measures. Our experiments show that the evaluated methods improve the performance of a state-of-the-art summarization framework and strongly indicate the benefits of continuous word vector representations for automatic summarization.

1 Introduction

The goal of summarization is to capture the important information contained in large volumes of text, and present it in a brief, representative, and consistent summary. A well written summary can significantly reduce the amount of work needed to digest large amounts of text on a given topic. The creation of summaries is currently a task best handled by humans. However, with the explosion of available textual data, it is no longer financially possible, or feasible, to produce all types of summaries by hand. This is especially true if the subject matter has a narrow base of interest, either due to the number of potential readers or the duration during which it is of general interest. A summary describing the events of World War II might for instance be justified to create manually, while a summary of all reviews and comments regarding a certain version of Windows might not. In such cases, automatic summarization is a way forward.

In this paper we introduce a novel application of continuous vector representations to the problem of multi-document summarization. We evaluate different compositions for producing sentence representations based on two different word embeddings on a standard dataset using the ROUGE evaluation measures. Our experiments show that the evaluated methods improve the performance of a state-of-the-art summarization framework which strongly indicate the benefits of continuous word vector representations for this tasks.

2 Summarization

There are two major types of automatic summarization techniques, extractive and abstractive. *Extractive summarization* systems create summaries using representative sentences chosen from the input while *abstractive summarization* creates new sentences and is generally considered a more difficult problem.

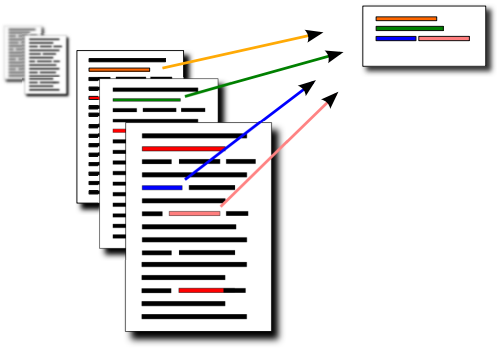

Figure 1: Illustration of Extractive Multi-Document Summarization.

For this paper we consider extractive multi-document summarization, that is, sentences are chosen for inclusion in a summary from a set of documents D. Typically, extractive summarization techniques can be divided into two components, the summarization framework and the similarity measures used to compare sentences. Next

we present the algorithm used for the framework and in Sec. 2.2 we discuss a typical sentence similarity measure, later to be used as a baseline.

2.1 Submodular Optimization

Lin and Bilmes (2011) formulated the problem of extractive summarization as an optimization problem using monotone nondecreasing submodular set functions. A submodular function F on the set of sentences V satisfies the following property: for any $A \subseteq B \subseteq V \backslash \{v\}$, $F(A+\{v\}) - F(A) \geq F(B+\{v\}) - F(B)$ where $v \in V$. This is called the diminishing returns property and captures the intuition that adding a sentence to a small set of sentences (i.e., summary) makes a greater contribution than adding a sentence to a larger set. The aim is then to find a summary that maximizes diversity of the sentences and the coverage of the input text. This objective function can be formulated as follows:

$$\mathcal{F}(S) = \mathcal{L}(S) + \lambda \mathcal{R}(S)$$

where S is the summary, $\mathcal{L}(S)$ is the coverage of the input text, $\mathcal{R}(S)$ is a diversity reward function. The λ is a trade-off coefficient that allows us to define the importance of coverage versus diversity of the summary. In general, this kind of optimization problem is NP-hard, however, if the objective function is submodular there is a fast scalable algorithm that returns an approximation with a guarantee. In the work of Lin and Bilmes (2011) a simple submodular function is chosen:

$$\mathcal{L}(S) = \sum_{i \in V} \min\{\sum_{j \in S} \mathrm{Sim}(i,j), \alpha \sum_{j \in V} \mathrm{Sim}(i,j)\} \tag{1}$$

The first argument measures similarity between sentence i and the summary S, while the second argument measures similarity between sentence i and the rest of the input V. $\mathrm{Sim}(i,j)$ is the similarity between sentence i and sentence j and $0 \leq \alpha \leq 1$ is a threshold coefficient. The diversity reward function $\mathcal{R}(S)$ can be found in (Lin and Bilmes, 2011).

2.2 Traditional Similarity Measure

Central to most extractive summarization systems is the use of sentence similarity measures ($\mathrm{Sim}(i,j)$ in Eq. 1). Lin and Bilmes measure similarity between sentences by representing each sentence using *tf-idf* (Salton and McGill, 1986) vectors and measuring the cosine angle between vectors. Each sentence is represented by a word vector $\mathbf{w} = (w_1, \ldots, w_N)$ where N is the size of the vocabulary. Weights w_{ki} correspond to the *tf-idf* value of word k in the sentence i. The weights $\mathrm{Sim}(i,j)$ used in the $\mathcal{L}$ function in Eq. 1 are found using the following similarity measure.

$$\mathrm{Sim}(i,j) = \frac{\sum_{w \in i} \mathrm{tf}_{w,i} \times \mathrm{tf}_{w,j} \times \mathrm{idf}_w^2}{\sqrt{\sum_{w \in i} \mathrm{tf}_{w,i}^2 \times \mathrm{idf}_w^2} \sqrt{\sum_{w \in j} \mathrm{tf}_{w,j}^2 \times \mathrm{idf}_w^2}} \tag{2}$$

where $\mathrm{tf}_{w,i}$ and $\mathrm{tf}_{w,j}$ are the number of occurrences of w in sentence i and j, and idf_w is the inverse document frequency (*idf*) of w.

In order to have a high similarity between sentences using the above measure, two sentences must have an overlap of highly scored *tf-idf* words. The overlap must be exact to count towards the similarity, e.g, the terms *The US President* and *Barack Obama* in different sentences will not add towards the similarity of the sentences. To capture deeper similarity, in this paper we will investigate the use of continuous vector representations for measuring similarity between sentences. In the next sections we will describe the basics needed for creating continuous vector representations and methods used to create sentence representations that can be used to measure sentence similarity.

3 Background on Deep Learning

Deep learning (Hinton et al., 2006) is a modern interpretation of artificial neural networks (ANN), with an emphasis on deep network architectures. Deep learning can be used for challenging problems like image and speech recognition (Krizhevsky et al., 2012; Graves et al., 2013), as well as language modeling (Mikolov et al., 2010), and in all cases, able to achieve state-of-the-art results.

Inspired by the brain, ANNs use a neuron-like construction as their primary computational unit. The behavior of a neuron is entirely controlled by its input weights. Hence, the weights are where the information learned by the neuron is stored. More precisely the output of a neuron is computed as the weighted sum of its inputs, and squeezed into the interval $[0,1]$ using a sigmoid function:

$$y_i = g(\theta_i^T \mathbf{x}) \tag{3}$$

$$g(z) = \frac{1}{1+e^{-z}} \tag{4}$$

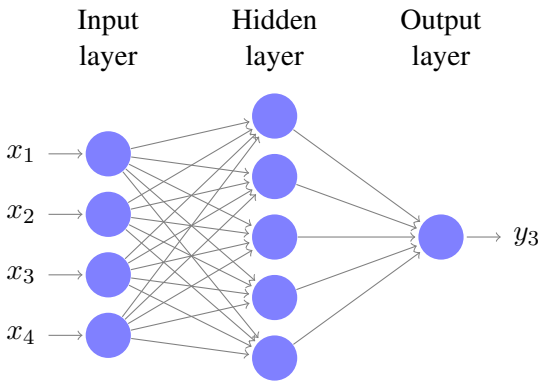

Figure 2: FFNN with four input neurons, one hidden layer, and 1 output neuron. This type of architecture is appropriate for binary classification of some data $\mathbf{x} \in \mathbb{R}^4$, however depending on the complexity of the input, the number and size of the hidden layers should be scaled accordingly.

where θ_i are the weights associated with neuron i and $\mathbf{x}$ is the input. Here the sigmoid function (g) is chosen to be the logistic function, but it may also be modeled using other sigmoid shaped functions, e.g. the hyperbolic tangent function.

The neurons can be organized in many different ways. In some architectures, loops are permitted. These are referred to as *recurrent neural networks*. However, all networks considered here are non-cyclic topologies. In the rest of this section we discuss a few general architectures in more detail, which will later be employed in the evaluated models.

3.1 Feed Forward Neural Network

A feed forward neural network (FFNN) (Haykin, 2009) is a type of ANN where the neurons are structured in layers, and only connections to subsequent layers are allowed, see Figure 2. The algorithm is similar to logistic regression using non-linear terms. However, it does not rely on the user to choose the non-linear terms needed to fit the data, making it more adaptable to changing datasets. The first layer in a FFNN is called the input layer, the last layer is called the output layer, and the interim layers are called hidden layers. The hidden layers are optional but necessary to fit complex patterns.

Training is achieved by minimizing the network error (E). How E is defined differs between different network architectures, but is in general a differentiable function of the produced output and

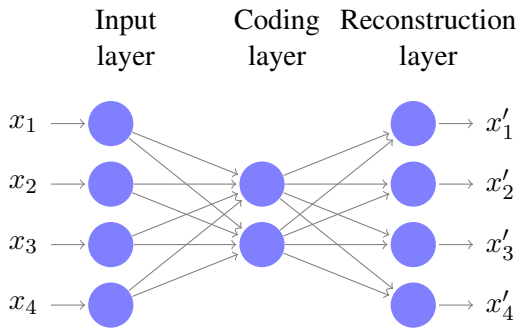

Figure 3: The figure shows an auto-encoder that compresses four dimensional data into a two dimensional code. This is achieved by using a bottleneck layer, referred to as a coding layer.

the expected output. In order to minimize this function the gradient $\frac{\partial E}{\partial \Theta}$ first needs to be calculated, where Θ is a matrix of all parameters, or weights, in the network. This is achieved using backpropagation (Rumelhart et al., 1986). Secondly, these gradients are used to minimize E using e.g. gradient descent. The result of this processes is a set of weights that enables the network to do the desired input-output mapping, as defined by the training data.

3.2 Auto-Encoder

An auto-encoder (AE) (Hinton and Salakhutdinov, 2006), see Figure 3, is a type of FFNN with a topology designed for dimensionality reduction. The input and the output layers in an AE are identical, and there is at least one hidden bottleneck layer that is referred to as the coding layer. The network is trained to reconstruct the input data, and if it succeeds this implies that all information in the data is necessarily contained in the compressed representation of the coding layer.

A shallow AE, i.e. an AE with no extra hidden layers, will produce a similar code as principal component analysis. However, if more layers are added, before and after the coding layer, non-linear manifolds can be found. This enables the network to compress complex data, with minimal loss of information.

3.3 Recursive Neural Network

A recursive neural network (RvNN), see Figure 4, first presented by Socher et al. (2010), is a type of feed forward neural network that can process data through an arbitrary binary tree structure, e.g. a

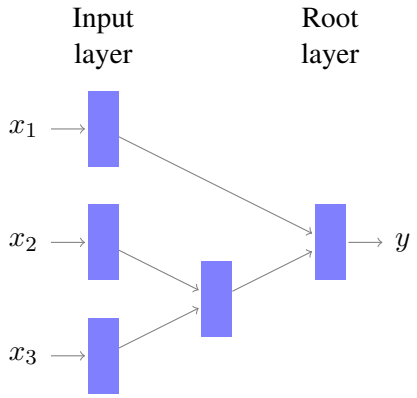

Figure 4: The recursive neural network architecture makes it possible to handle variable length input data. By using the same dimensionality for all layers, arbitrary binary tree structures can be recursively processed.

binary parse tree produced by linguistic parsing of a sentence. This is achieved by enforcing weight constraints across all nodes and restricting the output of each node to have the same dimensionality as its children.

The input data is placed in the leaf nodes of the tree, and the structure of this tree is used to guide the recursion up to the root node. A compressed representation is calculated recursively at each non-terminal node in the tree, using the same weight matrix at each node. More precisely, the following formulas can be used:

$$z_p = \theta_p^T [x_l; x_r] \tag{5a}$$

$$y_p = g(z_p) \tag{5b}$$

where y_p is the computed parent state of neuron p, and z_p the induced field for the same neuron. $[x_l; x_r]$ is the concatenation of the state belonging to the right and left sibling nodes. This process results in a fixed length representation for hierarchical data of arbitrary length. Training of the model is done using backpropagation through structure, introduced by Goller and Kuchler (1996).

4 Word Embeddings

Continuous distributed vector representation of words, also referred to as word embeddings, was first introduced by Bengio et al. (2003). A word embedding is a continuous vector representation that captures semantic and syntactic information about a word. These representations can be used to unveil dimensions of similarity between words, e.g. singular or plural.

4.1 Collobert & Weston

Collobert and Weston (2008) introduce an efficient method for computing word embeddings, in this work referred to as *CW* vectors. This is achieved firstly, by scoring a valid n-gram ($\mathbf{x}$) and a corrupted n-gram ($\bar{\mathbf{x}}$) (where the center word has been randomly chosen), and secondly, by training the network to distinguish between these two n-grams. This is done by minimizing the hinge loss

$$\max(0, 1 - s(\mathbf{x}) + s(\bar{\mathbf{x}})) \tag{6}$$

where s is the scoring function, i.e. the output of a FFNN that maps between the word embeddings of an n-gram to a real valued score. Both the parameters of the scoring function and the word embeddings are learned in parallel using backpropagation.

4.2 Continuous Skip-gram

A second method for computing word embeddings is the Continuous Skip-gram model, see Figure 5, introduced by Mikolov et al. (2013a). This model is used in the implementation of their word embeddings tool *Word2Vec*. The model is trained to predict the context surrounding a given word. This is accomplished by maximizing the objective function

$$\frac{1}{T} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{t+j} | w_t) \tag{7}$$

where T is the number of words in the training set, and c is the length of the training context. The probability $p(w_{t+j}|w_t)$ is approximated using the hierarchical softmax introduced by Bengio et al. (2002).

5 Phrase Embeddings

Word embeddings have proven useful in many natural language processing (NLP) tasks. For summarization, however, sentences need to be compared. In this section we present two different methods for deriving phrase embeddings, which in Section 5.3 will be used to compute sentence to sentence similarities.

5.1 Vector addition

The simplest way to represent a sentence is to consider it as the sum of all words without regarding word orders. This was considered by Mikolov et al. (2013b) for representing short

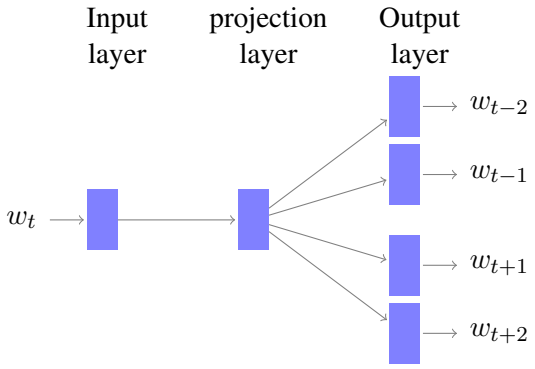

Figure 5: The continuous Skip-gram model. Using the input word (w_t) the model tries to predict which words will be in its context ($w_{t\pm c}$).

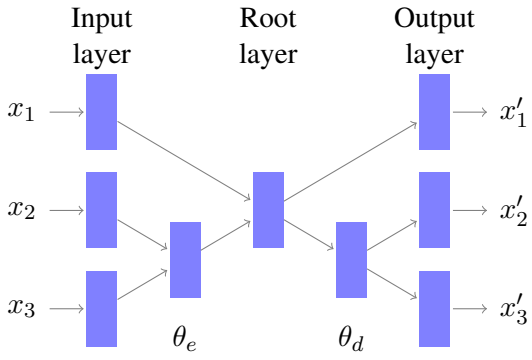

Figure 6: The structure of an unfolding RAE, on a three word phrase ($[x_1, x_2, x_3]$). The weight matrix θ_e is used to encode the compressed representations, while θ_d is used to decode the representations and reconstruct the sentence.

phrases. The model is expressed by the following equation:

$$\mathbf{x}_p = \sum_{\mathbf{x}_w \in \{\text{sentence}\}} \mathbf{x}_w \tag{8}$$

where x_p is a phrase embedding, and x_w is a word embedding. We use this method for computing phrase embeddings as a baseline in our experiments.

5.2 Unfolding Recursive Auto-encoder

The second model is more sophisticated, taking into account also the order of the words and the grammar used. An unfolding recursive auto-encoder (RAE) is used to derive the phrase embedding on the basis of a binary parse tree. The unfolding RAE was introduced by Socher et al. (2011) and uses two RvNNs, one for encoding the compressed representations, and one for decoding them to recover the original sentence, see Figure 6. The network is subsequently trained by minimizing the reconstruction error.

Forward propagation in the network is done by recursively applying Eq. 5a and 5b for each triplet in the tree in two phases. First, starting at the center node (root of the tree) and recursively pulling the data from the input. Second, again starting at the center node, recursively pushing the data towards the output. Backpropagation is done in a similar manner using backpropagation through structure (Goller and Kuchler, 1996).

5.3 Measuring Similarity

Phrase embeddings provide semantically aware representations for sentences. For summarization, we need to measure the similarity between two representations and will make use of the following two vector similarity measures. The first similarity measure is the cosine similarity, transformed to the interval of $[0, 1]$

$$\text{Sim}(i,j) = \left(\frac{\mathbf{x}_i^T \mathbf{x}_j}{\|\mathbf{x}_j\| \|\mathbf{x}_j\|} + 1 \right) /2 \tag{9}$$

where $\mathbf{x}$ denotes a phrase embedding The second similarity is based on the complement of the Euclidean distance and computed as:

$$\text{Sim}(i,j) = 1 - \frac{1}{\max\limits_{k,n} \sqrt{\| \mathbf{x}_k - \mathbf{x}_n \|^2}} \sqrt{\| \mathbf{x}_j - \mathbf{x}_i \|^2} \tag{10}$$

6 Experiments

In order to evaluate phrase embeddings for summarization we conduct several experiments and compare different phrase embeddings with *tf-idf* based vectors.

6.1 Experimental Settings

Seven different configuration were evaluated. The first configuration provides us with a baseline and is denoted *Original* for the Lin-Bilmes method described in Sec. 2.1. The remaining configurations comprise selected combinations of word embeddings, phrase embeddings, and similarity measures.

The first group of configurations are based on vector addition using both Word2Vec and CW vectors. These vectors are subsequently compared using both cosine similarity and Euclidean distance.

The second group of configurations are built upon recursive auto-encoders using CW vectors and are also compared using cosine similarity as well as Euclidean distance.

The methods are named according to: VectorType_EmbeddingMethod$_{SimilarityMethod}$, e.g. W2V_Add$_{Cos}$ for Word2Vec vectors combined using vector addition and compared using cosine similarity.

To get an upper bound for each ROUGE score, we use an exhaustive search on summaries. We evaluated each possible pair of sentences and maximized w.r.t the ROUGE score.

6.2 Dataset and Evaluation

The Opinosis dataset (Ganesan et al., 2010) consists of short user reviews in 51 different topics. Each of these topics contains between 50 and 575 sentences and are a collection of user reviews made by different authors about a certain characteristic of a hotel, car or a product (e.g. *"Location of Holiday Inn, London"* and *"Fonts, Amazon Kindle"*). The dataset is well suited for multi-document summarization (each sentence is considered its own document), and includes between 4 and 5 gold-standard summaries (not sentences chosen from the documents) created by human authors for each topic.

Each summary is evaluated with ROUGE, that works by counting word overlaps between generated summaries and gold standard summaries. Our results include R-1, R-2, and R-SU4, which counts matches in unigrams, bigrams, and skip-bigrams respectively. The skip-bigrams allow four words in between (Lin, 2004).

The measures reported are recall (R), precision (P), and F-score (F), computed for each topic individually and averaged. Recall measures what fraction of a human created gold standard summaries that is captured, and precision measures what fraction of the generated summary that is in the gold standard. F-score is a standard way to combine recall and precision, computed as $F = 2\frac{P*R}{P+R}$.

6.3 Implementation

All results were obtained by running an implementation of Lin-Bilmes submodular optimization summarizer, as described in Sec. 2.1. Also, we have chosen to fix the length of the summaries to two sentences because the length of the gold-standard summaries are typically around two sentences. The CW vectors used were trained by Turian et al. (2010)[1], and the Word2Vec vectors by Mikolov et al. (2013b)[2]. The unfolding RAE used is based on the implementation by Socher et al. (2011)[3], and the parse trees for guiding the recursion was generated using the Stanford Parser (Klein and Manning, 2003)[4].

6.4 Results

The results from the ROUGE evaluation are compiled in Table 1. We find for all measures (recall, precision, and F-score), that the phrase embeddings outperform the original Lin-Bilmes. For recall, we find that CW_Add$_{Cos}$ achieves the highest result, while for precision and F-score the CW_Add$_{Euc}$ perform best. These results are consistent for all versions of ROUGE scores reported (1, 2 and SU4), providing a strong indication for phrase embeddings in the context of automatic summarization.

Unfolding RAE on CW vectors and vector addition on W2V vectors gave comparable results w.r.t. each other, generally performing better than original Linn-Bilmes but not performing as well as vector addition of CW vectors.

The results denoted OPT in Table 1 describe the upper bound score, where each row represents optimal recall and F-score respectively. The best results are achieved for R-1 with a maximum recall of 57.86%. This is a consequence of hand created gold standard summaries used in the evaluation, that is, we cannot achieve full recall or F-score when the sentences in the gold standard summaries are not taken from the underlying documents and thus, they can never be fully matched using extractive summarization. R-2 and SU4 have lower maximum recall and F-score, with 22.9% and 29.5% respectively.

6.5 Discussion

The results of this paper show great potential for employing word and phrase embeddings in summarization. We believe that by using embeddings we move towards more semantically aware summarization systems. In the future, we anticipate improvements for the field of automatic summarization (as well as for similar applications), as the quality of the word vectors improves and we find

[1] http://metaoptimize.com/projects/wordreprs/

[2] https://code.google.com/p/word2vec/

[3] http://nlp.stanford.edu/ socherr/codeRAEVectorsNIPS2011.zip

[4] http://nlp.stanford.edu/software/lex-parser.shtml

Table 1: ROUGE scores for summaries using different similarity measures. OPT constitutes the optimal ROUGE scores on this dataset.

ROUGE-1			
	R	**P**	**F**
OPT_R	**57.86**	21.96	30.28
OPT_F	45.93	48.84	**46.57**
CW_RAE_{Cos}	27.37	19.89	22.00
CW_RAE_{Euc}	29.25	19.77	22.62
CW_Add_{Cos}	**34.72**	11.75	17.16
CW_Add_{Euc}	29.12	**22.75**	**24.88**
W2V_Add_{Cos}	30.86	16.81	20.93
W2V_Add_{Euc}	28.71	16.67	20.75
Original	25.82	19.58	20.57
ROUGE-2			
	R	**P**	**F**
OPT_R	**22.96**	12.31	15.33
OPT_F	20.42	19.94	**19.49**
CW_RAE_{Cos}	4.68	3.18	3.58
CW_RAE_{Euc}	4.82	3.24	3.67
CW_Add_{Cos}	**5.89**	1.81	2.71
CW_Add_{Euc}	5.12	**3.60**	**4.10**
W2V_Add_{Cos}	5.71	3.08	3.82
W2V_Add_{Euc}	3.86	1.95	2.54
Original	3.92	2.50	2.87
ROUGE-SU4			
	R	**P**	**F**
OPT_R	**29.50**	13.53	17.70
OPT_F	23.17	26.50	**23.70**
CW_RAE_{Cos}	9.61	6.23	6.95
CW_RAE_{Euc}	9.95	6.17	7.04
CW_Add_{Cos}	**12.38**	3.27	5.03
CW_Add_{Euc}	10.54	**7.59**	**8.35**
W2V_Add_{Cos}	11.94	5.52	7.12
W2V_Add_{Euc}	9.78	4.69	6.15
Original	9.15	6.74	6.73

enhanced ways of composing and comparing the vectors.

It is interesting to compare the results from different composition techniques on the CW vectors, where vector addition surprisingly outperforms the considerably more sophisticated unfolding RAE. However, since the unfolding RAE uses syntactic information in the text, this may be a result of using a dataset consisting of low quality text.

In the interest of comparing word embeddings, results using vector addition and cosine similarity were computed based on both CW and Word2Vec vectors. Supported by the achieved results CW vectors seems better suited for sentence similarities in this setting.

An issue we encountered with using precomputed word embeddings was their limited vocabulary, in particular missing uncommon (or common incorrect) spellings. This problem is particularly pronounced on the evaluated Opinosis dataset, since the text is of low quality. Future work is to train word embeddings on a dataset used for summarization to better capture the specific semantics and vocabulary.

The optimal R-1 scores are higher than R-2 and SU4 (see Table 1) most likely because the score ignores word order and considers each sentence as a set of words. We come closest to the optimal score for R-1, where we achieve 60% of maximal recall and 49% of F-score. Future work is to investigate why we achieve a much lower recall and F-score for the other ROUGE scores.

Our results suggest that the phrase embeddings capture the kind of information that is needed for the summarization task. The embeddings are the underpinnings of the decisions on which sentences that are representative of the whole input text, and which sentences that would be redundant when combined in a summary. However, the fact that we at most achieve 60% of maximal recall suggests that the phrase embeddings are not complete w.r.t summarization and might benefit from being combined with other similarity measures that can capture complementary information, for example using multiple kernel learning.

7 Related Work

To the best of our knowledge, continuous vector space models have not previously been used in summarization tasks. Therefore, we split this section in two, handling summarization and continuous vector space models separately.

7.1 Continuous Vector Space Models

Continuous distributed vector representation of words was first introduced by Bengio et al. (2003). They employ a FFNN, using a window of words

as input, and train the model to predict the next word. This is computed using a big softmax layer that calculate the probabilities for each word in the vocabulary. This type of exhaustive estimation is necessary in some NLP applications, but makes the model heavy to train.

If the sole purpose of the model is to derive word embeddings this can be exploited by using a much lighter output layer. This was suggested by Collobert and Weston (2008), which swapped the heavy softmax against a hinge loss function. The model works by scoring a set of consecutive words, distorting one of the words, scoring the distorted set, and finally training the network to give the correct set a higher score.

Taking the lighter concept even further, Mikolov et al. (2013a) introduced a model called Continuous Skip-gram. This model is trained to predict the context surrounding a given word using a shallow neural network. The model is less aware of the order of words, than the previously mentioned models, but can be trained efficiently on considerably larger datasets.

An early attempt at merging word representations into representations for phrases and sentences is introduced by Socher et al. (2010). The authors present a recursive neural network architecture (RvNN) that is able to jointly learn parsing and phrase/sentence representation. Though not able to achieve state-of-the-art results, the method provides an interesting path forward. The model uses one neural network to derive all merged representations, applied recursively in a binary parse tree. This makes the model fast and easy to train but requires labeled data for training.

The problem of needing labeled training data is remedied by Socher et al. (2011), where the RvNN model is adapted to be used as an auto-encoder and employed for paraphrase detection. The approach uses a precomputed binary parse tree to guide the recursion. The unsupervised nature of this setup makes it possible to train on large amounts of data, e.g., the complete English Wikipedia.

7.2 Summarization Techniques

Radev et al. (2004) pioneered the use of cluster centroids in their work with the idea to group, in the same cluster, those sentences which are highly similar to each other, thus generating a number of clusters. To measure the similarity between a pair of sentences, the authors use the cosine similarity measure where sentences are represented as weighted vectors of *tf-idf* terms. Once sentences are clustered, sentence selection is performed by selecting a subset of sentences from each cluster.

In TextRank (2004), a document is represented as a graph where each sentence is denoted by a vertex and pairwise similarities between sentences are represented by edges with a weight corresponding to the similarity between the sentences. The Google PageRank ranking algorithm is used to estimate the importance of different sentences and the most important sentences are chosen for inclusion in the summary.

Bonzanini, Martinez, Roelleke (2013) presented an algorithm that starts with the set of all sentences in the summary and then iteratively chooses sentences that are unimportant and removes them. The sentence removal algorithm obtained good results on the Opinosis dataset, in particular w.r.t F-scores.

We have chosen to compare our work with that of Lin and Bilmes (2011), described in Sec. 2.1. Future work is to make an exhaustive comparison using a larger set similarity measures and summarization frameworks.

8 Conclusions

We investigated the effects of using phrase embeddings for summarization, and showed that these can significantly improve the performance of the state-of-the-art summarization method introduced by Lin and Bilmes in (2011). Two implementations of word vectors and two different approaches for composition where evaluated. All investigated combinations improved the original Lin-Bilmes approach (using *tf-idf* representations of sentences) for at least two ROUGE scores, and top results where found using vector addition on CW vectors.

In order to further investigate the applicability of continuous vector representations for summarization, in future work we plan to try other summarization methods. In particular we will use a method based on multiple kernel learning where phrase embeddings can be combined with other similarity measures. Furthermore, we aim to use a novel method for sentence representation similar to the RAE using multiplicative connections controlled by the local context in the sentence.

References

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155.

Yoshua Bengio. 2002. New distributed probabilistic language models. Technical Report 1215, Département d'informatique et recherche opérationnelle, Université de Montréal.

Marco Bonzanini, Miguel Martinez-Alvarez, and Thomas Roelleke. 2013. Extractive summarisation via sentence removal: Condensing relevant sentences into a short summary. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '13, pages 893–896. ACM.

Ronan Collobert and Jason Weston. 2008. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM.

Kavita Ganesan, ChengXiang Zhai, and Jiawei Han. 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. ACL.

Christoph Goller and Andreas Kuchler. 1996. Learning task-dependent distributed representations by backpropagation through structure. In *IEEE International Conference on Neural Networks*, volume 1, pages 347–352. IEEE.

Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. Speech recognition with deep recurrent neural networks. *arXiv preprint arXiv:1303.5778*.

S.S. Haykin. 2009. *Neural Networks and Learning Machines*. Number v. 10 in Neural networks and learning machines. Prentice Hall.

Geoffrey E Hinton and Ruslan R Salakhutdinov. 2006. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507.

Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. 2006. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554.

Dan Klein and Christopher D Manning. 2003. Fast exact inference with a factored model for natural language parsing. *Advances in neural information processing systems*, pages 3–10.

Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1106–1114.

Hui Lin and Jeff Bilmes. 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*, pages 510–520. ACL.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*, pages 74–81.

Rada Mihalcea and Paul Tarau. 2004. TextRank: Bringing order into texts. In *Proceedings of EMNLP*, volume 4. Barcelona, Spain.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *INTERSPEECH*, pages 1045–1048.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. *ArXiv preprint arXiv:1301.3781*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119.

Dragomir R Radev, Hongyan Jing, Małgorzata Styś, and Daniel Tam. 2004. Centroid-based summarization of multiple documents. *Information Processing & Management*, 40(6):919–938.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1986. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Gerard Salton and Michael J. McGill. 1986. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., New York, NY, USA.

Richard Socher, Christopher D Manning, and Andrew Y Ng. 2010. Learning continuous phrase representations and syntactic parsing with recursive neural networks. In *Proceedings of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop*.

Richard Socher, Eric H. Huang, Jeffrey Pennington, Andrew Y. Ng, and Christopher D. Manning. 2011. Dynamic Pooling and Unfolding Recursive Autoencoders for Paraphrase Detection. In *Advances in Neural Information Processing Systems 24*.

Joseph Turian, Lev Ratinov, and Yoshua Bengio. 2010. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 384–394. ACL.