ELEC-E5550 - Statistical Natural Language Processing, Project Report: Comparison of word2vec Embeddings in Sentiment Analysis

Jarkko Tulensalo (84071T), Julius Hietala (317447)

April 2020

1 Introduction

Representing words as vectors is essential to almost any machine learning system that uses words as inputs. Many simple approaches such as tf-idf [14] based vectors have been used for a long time. One shortcoming of such simple methods is that there is not a notion of similarity or other relationships between different words. However, more recently more sophisticated methods such as word2vec [8] have emerged as the de facto approaches for embedding words into vectors, since they seem to express similarity and other relationships between words. In this work, we experiment with training different word2vec vectors with different hyperparameter settings and analyze them by using the metrics provided by [8]. We apply the learnt word embeddings as input in an important natural language processing task, sentiment analysis, where the target is to distinguish if the sentiment of a given text is positive or negative. Since we have trained the embeddings in an unsupervised manner, the performance metrics available do not directly apply to the sentiment analysis task since that is done in a supervised manner. The main question we try to answer is that whether performance metrics in the different tasks correlate with each other. As a dataset, we used the Sentiment140 dataset [4] that contains labeled (positive/negative) Twitter tweets and as the sentiment analysis model we used a recurrent neural network (RNN) [3] model created using Pytorch [7]. In this report we will describe the methods, experiments, and results obtained during the project.

2 Methods

In this section, we first describe the background and theory behind word2vec, and then describe the background behind recurrent neural network (RNN) based text classification.

2.1 word2vec

word2vec was first introduced in [8] in 2013. The main idea is to go through the text corpus a word at a time, and train embeddings for each word type based on the contexts (other words appearing in a window around the word token) the tokens appear in. The paper introduced two different architectures to solve the problem: 1) continuous bag-of-words (CBOW) and 2) skip-gram. In the CBOW version, the idea is to try to maximize the probability of the target word, given the context words (usually the sum or the mean of them). On the other hand, in the skip-gram model the idea is to maximize the probability of the context words, given the target word. In practice these architectures can be implemented using 1-layer neural networks with no nonlinearity. The models are illustrated in Figure 1, where they take as inputs the current word embeddings (target or context based on the architecture), and the embeddings are then updated based on the loss function using gradient descent and backpropagation.

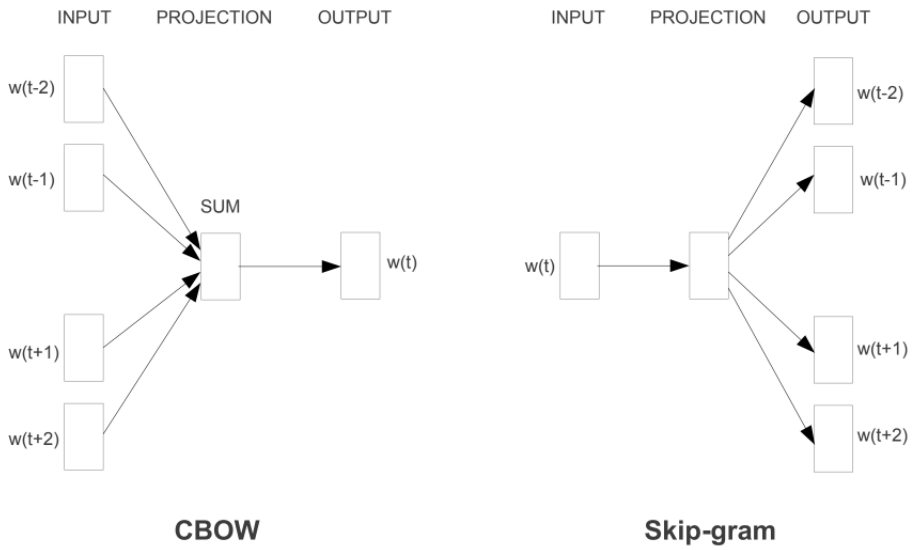

Figure 1: The CBOW and skip-gram architectures as 1-layer neural networks from [8]

The loss can be formulated as the softmax function

$$P(w_O|w_I) = \frac{exp(v_{w_O}^{'T} v_{w_I})}{\sum_{w=1}^{W} exp(v_w^{'T} v_{w_I})}$$

where w_O is the output word and $v_{w_O}^{'}$ is its embedding and similarly w_I is the input word and v_{w_I} its embedding. Each word w also has two representations $v_w^{'}$ and v_w that are used when the word is a part of the output or the input, respectively. This would be an attractive solution, but computing it would be prohibitively expensive since it would involve computations proportional to $|W|$ (the vocabulary size), which can easily be in the millions. Therefore [8] used hierarchical softmax [10] with binary Huffman tree as the vocabulary, which ensures that calculating the approximation of the softmax takes much less time for more frequent words. This approach worked well, but [9] proposed an approach called negative sampling, that works even better. In negative sampling (using the skip-gram as an example), the idea is to maximize the loss

$$L = \log \sigma(v_{w_O}^{'T} v_{w_I}) + \sum_{i=1}^{k} \log \sigma(-v_{w_i}^{'T} v_{w_I})$$

where k is the amount of negative samples. In other words, the idea is to try to teach the model to be able to recognize the true target-context pairs from

pairs where either the target or context (depending on architecture) is randomly drawn from some distribution. In the paper, they used the unigram distribution (unigram frequencies)raised to the power 0.75 for drawing the random samples. The example was provided for skip-gram for a single target-context word pair, but the same logic can be applied to the CBOW architecture as well.

2.2 Recurrent neural network

In recent literature, deep neural networks have shown high accuracy in natural language processing tasks. In this project report, we apply two bidirectional recurrent neural network models (RNN), Long Short-Term Memory [5] (LSTM) and GRU [1] for the chosen sentiment analysis task. Recurrent neural network models can learn dependencies between words for long and variable length sentences where traditional n-gram based models struggle. Recurrent neural network models also takes into account word order which is not possible for n-gram models. GRU is a simpler version of LSTM which was laterly introduced and shown decreased training time and has shown improved or similar performance in different applications.

Figure 2 shows the architecture of our proposed model. Each word in the given input sentence is converted into a word vector based on the learnt embeddings of the word2vec model. We applied a bidirectional RNN model which means that the vectors are given as an input to the RNN model where the words are fed from start to end for the first layer and in reverse order for the second layer. The RNN model updates the information between the cells at each step which are words in our case. The updated information is then given as an input to the next word. The final hidden state values of each cell are then combined with a linear layer which converts the learnt information into a single value which can be interpreted as positive or negative sentiment. We used sigmoid activation function to convert the value after the linear layer into a value between 0 and 1. Value over 0.5 was considered positive as value under 0.5 was considered negative.

3 Experiments

We first trained many sets (216) of different word2vec embeddings and analyzed them using the Semantic-Syntactic Word Relationship test set provided by [8]. Based on that performance metric, we chose the best performing set of vectors, and used those to train a sentiment analysis model and compared the accuracy to a baseline model. We also chose a set of vectors that did not perform so well in the Semantic-Syntactic Word Relationship test, and used those to train the sentiment analysis model to see whether the performance metrics in the two systems correlate.

As the data set, we chose the Sentiment140 dataset, that contains 1.6 Million sentiment labeled Twitter tweets. The distribution of the sentiment was even i.e. 50% of the tweets are positive and 50% were negative. One of the main

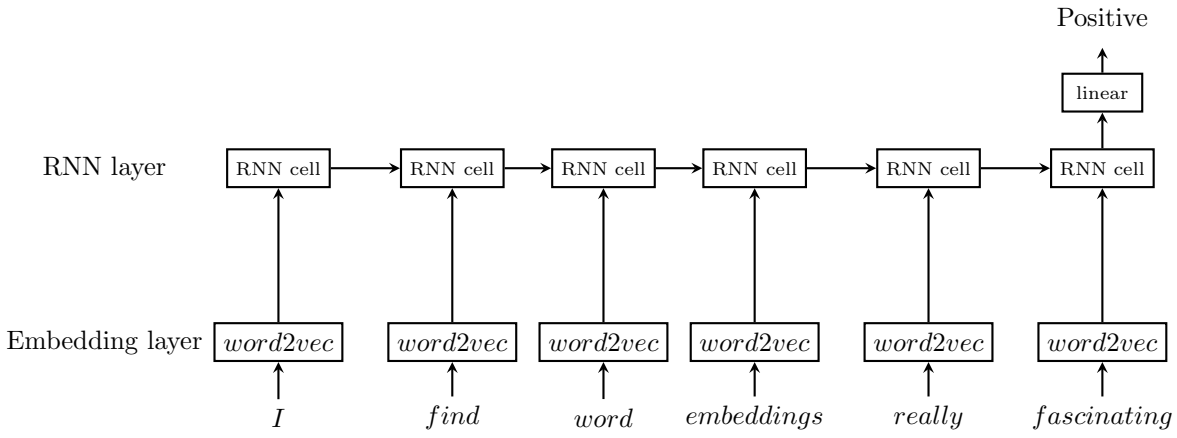

Figure 2: The recurrent neural network uses the learnt word embeddings from word2vec as an input. These word vectors are then fed into the RNN model. The last linear layer produces a prediction of the sentiment if the given sentence is positive or negative.

parameters experimented with was whether to include stop words i.e. the most commonly occurring words in the English language from the sentences or not. As shown in 1, we can see that the most frequent words in our dataset are words that might not have as much sentimental meaning by itself. However, we wanted to experiment if keeping these words can help the model to understand meaning of a whole sentence better.

We used the stop words provided by the nltk python package [11]. Figure 3 shows the length distributions of the sentence lengths in both cases. These distributions influenced some of the hyperparameter choices for training the word vectors as described below.

The dataset size (21 Million total words, vocabulary size 330k for the corpus including stop words) is low compared to the dataset used by [8], which might cause some implications when training the word2vec vectors. However given that we wanted to use the trained word vectors in the end task of sentiment analysis, we decided to use the same data set for both tasks.

Below, we will first describe the experiments performed when training the word2vec vectors, and training the sentiment analysis model, respectively.

3.1 word2vec

As mentioned above, the dataset size we used was low compared to [8]. However, in the paper the models were generally trained only for 1-3 passes through the whole data. To try to counteract the data size deficit, we trained our models for 10, 30, and 100 epochs.

The other major choice we tested was the model architecture. We used the

Table 1: 10 most frequent words in the vocabulary of the Sentiment140 dataset

Word	**Frequency**
i	613001
to	362478
the	334956
a	245300
my	202669
it	195707
and	194085
you	193059
is	151853
for	138680

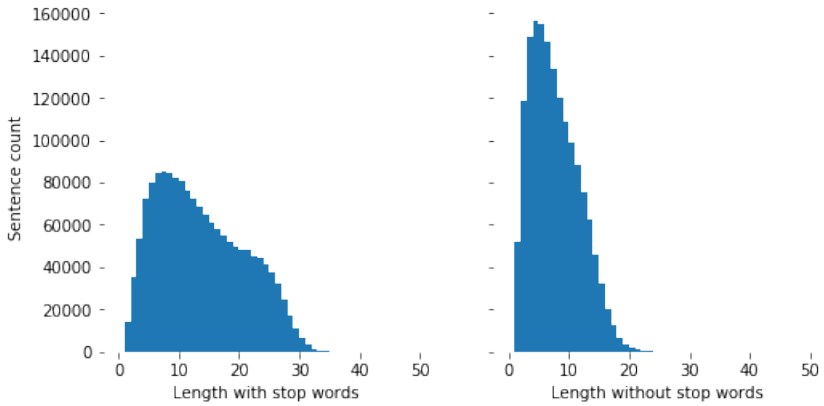

Figure 3: Length distribution of the sentences with and without stop words included

gensim python package [12] for training the word vectors, and it provided a simple way of switching between the CBOW and skip-gram architectures.

Other hyperparameters we tested were window size (the context size around the target word, maximum distance from the target word), including/removing stop words, vector embedding size, how many negative noise word examples to draw as introduced by [9].

Table 2 shows the tested hyperparameters for the word2vec vectors. The different window sizes when using stop words or not were motivated by the different sentence length distributions i.e. we wanted to compare cases where only a limited context around the target word is taken into account, to where a much longer total context is taken into account. The embedding sizes were motivated by [8], which mentions that embedding sizes under 100 tend not to work that well and their models were mainly using sizes 300 and 600. For the amount of noise word draws, [9] mentions that values 2-5 tend to work well

Table 2: Hyperparameters tested during word2vec training

Hyperparameter	**Experimented values**
Model	CBOW, Skip-gram
Maximum distance from the target	2, 7 (no stops), 8 (with stops)
Embedding size	100, 300, 600
Noise words	2, 5, 20
Stop words	yes, no
Epochs	10, 30, 100

enough for large datasets, whereas value 5-20 are required for smaller data sets.

To speed up training, embeddings were learned only for words whose minimum count in the full corpus was 3 or more. This procedure reduced the vocabulary size from around 330k to around 80k in both stop word inclusion settings. After the best model and hyperparameters were found, those were used then to train the model once again for also words with counts less than 3, since that would minimize the out-of-vocabulary (OOV) words during the sentiment analysis training and inference. The same procedure was done to the hyperparameter setting that produced less than optimal results that was used for comparison in the sentiment analysis task.

3.2 Sentiment analysis

We experimented if pretrained word2vec embeddings can improve the performance of our RNN architecture in the sentiment analysis task. We compared how a baseline model performs with three different embedding configurations:

- without any prelearnt embeddings,
- with frozen best performing word2vec embeddings and
- frozen suboptimal word2vec embeddings.

We ran the experiments related to using word2vec embeddings by freezing the weights of the embedding layer of the RNN. This means that the model does not update the embedding vectors that have been learnt by the word2vec model. For baseline, we allowed the model to optimise the embedding weights as well but they were initialised randomly. The baseline RNN model architecture was used for all three experiments. The only difference between the RNN models was how embedding layer was initialised and optimised during training.

We optimised the hyperparameters of the baseline model with a predefined grid search shown in 3. The optimised baseline architecture comprised of a one layer LSTM model with a hidden size of 256. Increasing the layers did not change the performance of the model but incereased training time which is why we chose the simplest model of 1 layer. LSTM showed slightly higher accuracy than GRU. We could see from the hyperparameter search that a dropout value 0.4 and larger batch size helped the model in generalisation. We used an embedding dimension

Table 3: Hyperparameters tuned for the recurrent neural network model.

Hyperparameter	**Experimented parameters**	**Chosen parameter**
Model	GRU, LSTM	LSTM
LSTM number of layers	1, 3	1
LSTM hidden size	128, 256	256
LSTM dropout	0.0, 0.2, 0.4, 0.5	0.4
Batch size	64, 128	128

of 300 for the baseline which matches the dimension of the best performing word2vec dimension.

We included unknown, start of sentence, end of sentence and padding tokens when processing sentences to the RNN model. We set the weights of the unknown and padding tokens to zero before training the model. We used adam optimiser and BCEWithLogitsLoss as the loss function of the model. We used accuracy of the model in the task as the performance metric. We ran each experiment a total of 10 epochs.

We split the data into a training - validation - test datasets with proportions of 70% - 20% - 10%. We used the validation dataset for the hyperparameter optimisation of the baseline model and test dataset for the performance comparison in the results section.

4 Results

4.1 word2vec

The Semantic-Syntactic Word Relationship test set was created by Mikolov et al. [8] and is available on the internet from `http://www.fit.vutbr.cz/~imikolov/rnnlm/word-test.v1.txt`. The data set has 5 semantic categories, and 9 syntactic categories, each containing word pair pairs. The idea is that by subtracting the second vector from the first in the first pair, and then adding the second vector to it from the second pair, the nearest neighbor to the resulting vector should be the first vector in the second pair. For example, vec("Helsinki") - vec("Finland") + vec("Sweden") should be closest to vec("Stockholm"). The test considers only these cases to be correct i.e. it does not take into account synonyms or other alternative "correct" forms. Thus one can expect that reaching 100 percent accuracy might be impossible. Figure 4 shows some examples of the data set.

We used the same test set and accuracy metric to rank the trained word vectors for being used in the sentiment analysis task. Since there were 216 different combinations, the full results are only provided in the appendix. To highlight the best parameter setting, only the top 10 settings based on accuracy are shown in Table 4. Only the pairs from the test set were used for which there were available embeddings in the trained vectors.

It can clearly be seen that the combination of CBOW, including stop words,

Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

Figure 4: Examples from the Semantic-Syntactic Word Relationship test set provided by [8]

and 30 epochs seems to produce the top results based on the Semantic-Syntactic Word Relationship accuracy metric. Also, using 5 or 20 noise words seems to work better than only 2. It is somewhat surprising that the CBOW model would produce the best results, since [8] reported that in their experiments the skip-gram performed better overall, especially in the semantic part. Including the stop words in the sentences also seems to improve the accuracy. Although some applications might benefit of discarding the redundant stop words, here it seems that they provide semantic and syntactic meaning. A thing to note is that the negative sampling already tries to account for the most frequent words, so removing them explicitly seems redundant. As mentioned, the original papers introducing and refining word2vec only train the models 1-3 epochs. It is somewhat unclear why in our case the 30 epochs would perform better than 10 or 100. When one trains with our dataset for 30 epochs, the model sees 630 Million target words, compared to more than 2 Billion target words when training with 100 epochs. In comparison, the model sees 1 Billion target words when trained for one epoch when using the dataset in [9]. Given the sentiment of favoring more data, one could maybe expect that the model that sees the most data would be the best. Clearly, that is not the case in our experiments, since 30 epochs is performing better than 100 epochs.

The best performing model was also trained again, but now with the full vocabulary i.e. also including words whose frequency was only 1. This slightly increased the accuracy to **0.22**.

It is also clear that the results are much worse than what reported by [9],

Table 4: Semantic-Syntactic Word Relationship accuracy results

Model	**Epochs**	**Stop words**	**Window**	**Size**	**Noise**	**Accuracy**
CBOW	30	yes	8	300	20	**0.18**
CBOW	30	yes	8	600	20	0.17
CBOW	30	yes	8	100	20	0.16
CBOW	30	yes	2	600	5	0.16
CBOW	30	yes	2	300	20	0.16
CBOW	30	yes	8	300	5	0.16
CBOW	30	yes	8	600	5	0.16
CBOW	30	yes	2	600	20	0.16
CBOW	30	yes	2	100	20	0.15
CBOW	30	yes	2	300	5	0.15

where the best model achieved 61% accuracy. This is most likely due to the fact that the amount of data in our experiments was much less. However [8] also ran tests with limited amounts of data to see how the data and embedding size affect the results. In one of the settings, they only used 24 Million words with embedding size 300 (and only a vocabulary size of 30k), and achieved an accuracy of 23.2%. This is somewhat in line with our results, since our corpus only had 21 Million words. So we expect that had we had more data available, the results would have been better.

Since there are 14 different semantic and syntactic categories, it is interesting to see in which ones the best model performed well, and in which ones it did not. Table 5 shows the results for the different categories. It is clear that some of the categories perform abysmally, whereas some categories perform fairly well. This might be due to the fact that Twitter tweets are quite informal i.e. they often do not follow proper grammar, so some of the details are not learned.

Based on the results, we also chose a suboptimally performing set of vectors, and retrained the model with the full vocabulary to get comparable results. A setting with CBOW, 10 epochs, including stop words, window size 8, vector size 600, and 2 noise words initially gave an accuracy of 0.05, and training with the full vocabulary did not improve this. We can see from the results that the main levers why this setting was worse than the best performing one were that the epochs were too low and there were too little noise words.

The Semantic-Syntactic Word Relationship test was the criterion based on which we chose the sets of vectors to the sentiment analysis task, whose results will be recapped below. However there are also other types performance tests available for the word2vec vectors. We defined 5 categories of words, 5 words in each category (listed in Table 6) and gathered the corresponding vectors from the best performing set of vectors, and the bad performing set vectors based on accuracy and performed principal component analysis (PCA) on them, and plotted the results into Figure 5. It is easy to see that in the better performing set of vectors, the categories form more clear and distant clusters when compared to the worse performing vectors.

Table 5: Results for different semantic and syntactic categories for the best performing model

Category	**Accuracy**
Common capital city	0.22
All capital cities	0.17
Currency	0.006
City-in-state	0.13
Family	0.52
Adjective to adverb	0.01
Opposite	0.02
Comparative	0.37
Superlative	0.13
Present participle	0.45
Nationality adjective	0.11
Past tense	0.34
Plural nouns	0.33
Plural verbs	0.07

Table 6: Words used to test different sets of embeddings

Category	**Words**
Studying	university, college, freshman, master, books
Colors	green, blue, red, black, yellow
Sports	hockey, basketball, baseball, football, tennis
Cities	shanghai, stockholm, helsinki, tokyo, london
Broad concepts	fashion, music, art, science, physics

Another way to estimate the performance of the models is to determine the nearest neighbor of a certain word, and to see whether is seems semantically similar. Table 7 shows the nearest neighbors for the first word from each category for both the best model and the suboptimal model. Based on those, it is hard to interpret which one is "better". Some of the nearest neighbors seem a bit off for even the best model, which hints that the embeddings are not that fine grained in the local neighborhoods of the words, but broader categories are still recognizable as shown in Figure 5.

4.2 Sentiment analysis

Table 8 shows the accuracy of the baseline RNN model compared to two models with pretrained word2vec embeddings. Firstly, the results show that we can use the proposed architecture to predict a sentiment of a tweet with fair accuracy. The RNN model that used best performing word2vec word embeddings also achieved highest accuracy of 83.3 % in the sentiment analysis task.

The results indicate that the RNN model can benefit in sentiment analysis from initialising and freezing the embedding layer with pretrained vectors with

Table 7: Nearest neighbors of categorical words for the different models

Best model	
Word	**Nearest neighbors**
university	faculty, college, colleges, professors, students
green	lantern, oolong, stripe, overlay, blue
hockey	penguins, nhl, redwings, playoff, canucs
shanghai	tedxshanghai, beijing, lufthansa, hk, qatar
fashion	philippine, fashiondelicious, depraved, tradeshow, snowboarder
Suboptimal model	
Word	**Nearest neighbors**
university	student, students, workshop, volunteering, seminar
green	rainbow, flower, silver, roses, blue
hockey	yankees, cubs, football, penguins, bulls
shanghai	northwest, cork, budapest, southwest, lewes
fashion	indie, blackfield, photograph, magazine, fundraisers

Table 8: Sentiment analysis accuracy of

Model	**Accuracy**
	%
Baseline	83.0
Best word2vec frozen	83.3
Low word2vec frozen	82.4

the help of an unsupervised word2vec algorithm. We can also observe that both models with pretrained word2vec embeddings did not outperform the baseline model. The difference between the model with suboptimal word2vec embeddings and baseline was only 0.5 %.

The results partly agree with a study on sentiment analysis with LSTM by Johnson Zhang. [6]. They explain that careful initilisation of the embedding layer might improve the model accuracy in a sentiment analysis task as we can see here with the best performing word2vec embeddings. We can also observe that the suboptimal word embeddings did not improve the model performance. However, in their study, they achieved higher accuracy without any pretrained vector initialisation which is similar to our baseline experiment. We found higher accuracy with the pretrained best word2vec vectors.

Our model results are comparable to a study by Saeidi et al. which showed sentiment analysis accuracy of 82 % with an LSTM based model architecture [13]. However, our model performs with lower accuracy than current state-of-the-art models. Devlin et al. reported close to 95 % accuracy in sentiment analysis task with a Transformers based deep learning architecture [2]. which suggests that with other deep learning architectures we could achieve even better accuracy in sentiment analysis task.

Figure 6 show the confusion matrices of the sentiment analysis for baseline

model and the model with the best perfoming word2vec pretrained embeddings. We can observe that the model with best performing pretrained embeddings can capture both positive and negative sentiments more evenly, where the baseline model has more difficulties in predicting positive sentiments. However, the differences are not substantial.

5 Discussion

In this report, we described our experiments training different word2vec embeddings. We also used a well known Semantic-Syntactic Word Relationship test set to measure the quality of the trained embeddings. Based on the results, we then chose one well performing and one suboptimally performing set of vectors to be used in a sentiment analysis task. The motivation for this was to see whether the two different performance metrics correlate. We found the most important factors to be the amount of noise words and amount of epochs when testing with the Semantic-Syntactic Word Relationship test set.

After training the embeddings and observing the results, we realized that the amount of data we were using was less than optimal, thus yelding results worse than what have been reported in research. We could still produce a wide range of sets of embeddings with different test performance, which kept the problem at hand still interesting.

We also showed that we can use the proposed recurrent neural network model to predict sentiments of twitter tweets. Our study indicates that initialising and freezing the embedding layer of the model with carefully pretrained word2vec vectors can improve the model prediction accuracy in a sentiment analysis task.

There are two aspects for further study in order to improve the model performance in the sentiment analysis task. Firstly, we can study if a bigger training dataset could improve the word2vec model in the unsupervised task performance and thereby also in the sentiment analysis task. Secondly, we can apply other deep learning models, which have shown higher performance in sentiment analysis than recurrent neural networks.

6 Division of labor

We contributed to the project with equal share. Collaboration was fluent and the group was happy with the process and results.

6.1 Julius Hietala (317447, julius.hietala@aalto.fi)

- Researching word2vec
- Researching implementation details, gensim
- Building training code for word2vec
- Running word2vec experiments

- 50% of report writing

6.2 Jarkko Tulensalo (84071T, jarkko.tulensalo@aalto.fi)

- Research on word embeddings and RNN models
- Researching implementation details on RNN models
- Implementing dataset preprocessing
- Implementing and running RNN hyperparameter search and experiments on sentiment analysis
- 50% of report writing

References

[1] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 10 2018.

[3] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. 1999.

[4] Alec Go, Richa Bhayani, and Lei Huang. Twitter sentiment classification using distant supervision. *CS224N project report, Stanford*, 1(12):2009, 2009.

[5] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[6] Rie Johnson and Tong Zhang. Supervised and semi-supervised text categorization using lstm for region embeddings, 2016.

[7] Nikhil Ketkar. Introduction to pytorch. In *Deep learning with python*, pages 195–208. Springer, 2017.

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

[9] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[10] Frederic Morin and Yoshua Bengio. Hierarchical probabilistic neural network language model. In *Aistats*, volume 5, pages 246–252. Citeseer, 2005.

[11] Nltk documentation. [Online; accessed 24-April-2020].

[12] Radim Řehřek and Petr Sojka. Gensim—statistical semantics in python. *Retrieved from genism. org*, 2011.

[13] Marzieh Saeidi, Guillaume Bouchard, Maria Liakata, and Sebastian Riedel. Sentihood: Targeted aspect based sentiment analysis dataset for urban neighbourhoods. 10 2016.

[14] Wikipedia contributors. Tf–idf — Wikipedia, the free encyclopedia, 2020. [Online; accessed 24-April-2020].

A Appendix

Code for the project available at: `https://github.com/hietalajulius/aalto-snlp-2020`
Full word2vec accuracy results: `https://docs.google.com/spreadsheets/d/1OPUHrJFdb4yxEXJvhOgMqVUZf_OhqYOEw0boq2fn3TU/edit?usp=sharing`

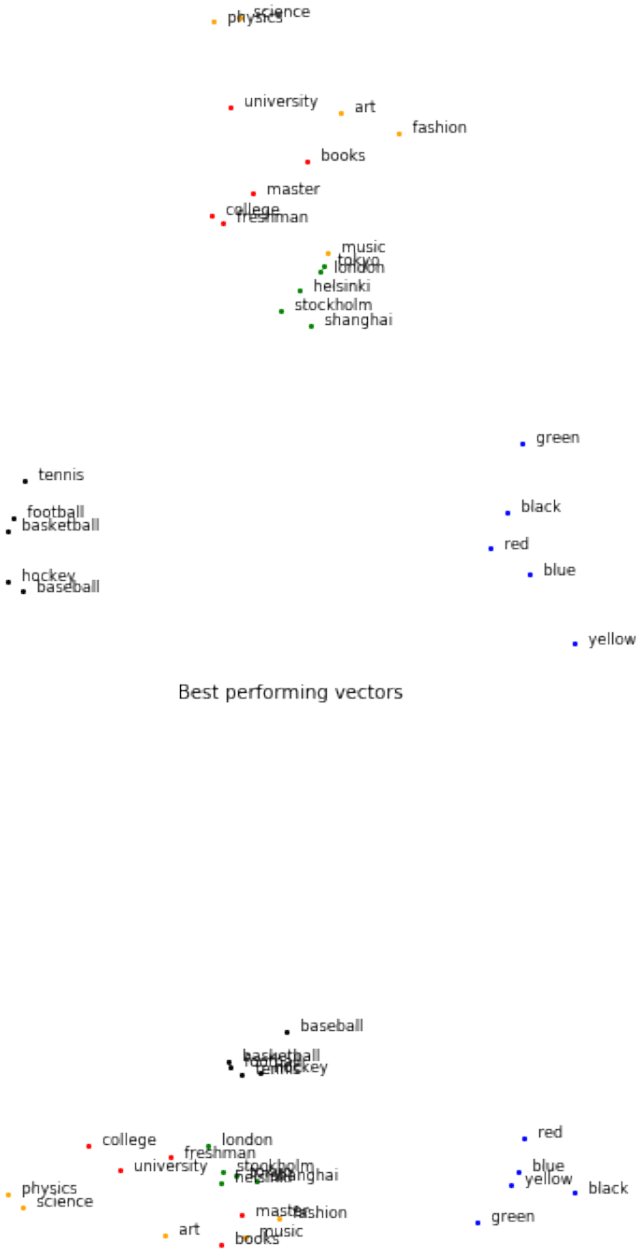

Figure 5: Dimension reduced word vectors for the best performing set and the suboptimally performing set

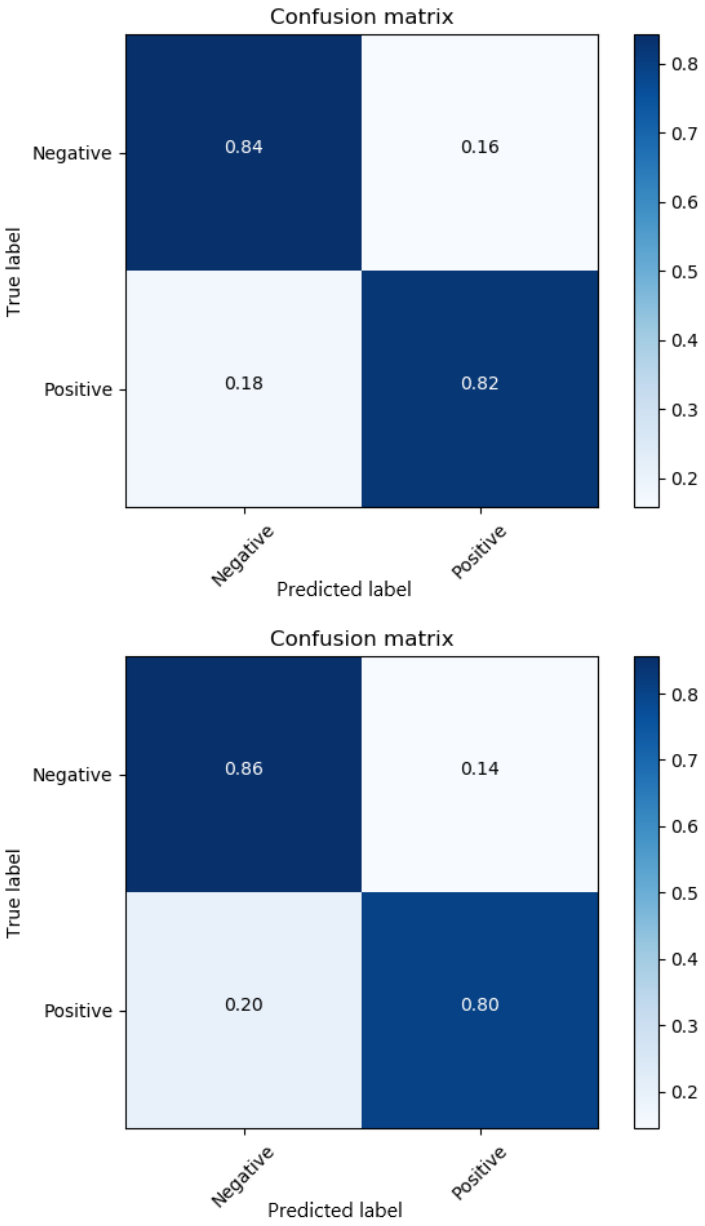

Figure 6: Confusion matrix of the sentiment analysis task with two experiments. Up: Using pretrained emebddings of the best performing word2vec model Down: Baseline model without pretrained word embedding initialisation