Implementation of Convolutional Neural Networks for Sentence Classification by Yoon Kim

Aixin Jiang
Faculty of Arts
McGill University
Montreal, Canada
aixin.jiang@mail.mcgill.ca

Yu-Cheng Cho
Department of Electrical and Computer Science
McGill University
Montreal, Canada
yu-cheng.cho@mail.mcgill.ca

Navpreet Singh
Department of Electrical and Computer Science
McGill University
Montreal, Canada
navpreet.singh@mail.mcgill.ca

***Abstract*— This report studies the baseline model proposed in the article "Convolutional Neural Networks for Sentence Classification" published by Yoon Kim in 2014. Two achievements are presented in this report. First, we demonstrate the successful reproducing of the baseline model utilizing the movie review dataset. An averaged accuracy of 80.71% is achieved by the authors (81.0% in the article). Second, we research on improving the reproduced baseline by conducting an extensive study on the hyper-parameters, including (1) the filter window sizes, (2) the number of feature maps, (3) activation functions, (4) dropout rates, (5) normalization constraints and (6) optimizers. Furthermore, the architecture of the convolutional neural network and the word embedding method are also studied. In conclusion, the prediction accuracy of 81.64% is achieved in this report.**

Keywords* — *CNN, word embedding, sentiment analysis, NLP

I. Introduction

For this project, we selected the paper Convolutional Neural Networks for Sentence Classification by Yoon Kim, published in 2014. The paper reports a series of experiments with convolutional neural networks (CNN) trained on top of pre-trained word vectors for sentence-level classification tasks. Kim (2014) proposed a simple one-layer CNN that achieved state of the art results across several datasets. This paper is one of the first implementations of CNN for sentence classification. Earlier, simple models like Logistic regression or SVM (Joachims, 1998) were a standard for text classification. This paper replaced these models, beating the performance in most of the benchmark datasets like movie reviews, Stanford sentiment treebank, subjectivity dataset, customer reviews, etc. The main idea of the paper is to replace each word of the text of the dataset with a vector representation of a fixed length. Kim implements different variants of the model and we fix one of these variants as our baseline and try to improve it. Since the author uses convolutional neural networks, we have several hyperparameters that can be tuned to get better accuracy than the paper. Hence, we started by tweaking the hyper-parameters of the model, followed by a slight modification to the architecture of the model.

The last step is to try to replace the word representation used by the author with some other publicly available word representations or their combinations. We were able to reproduce the results of the paper in one of the variants and improved the results for another variant.

II. Brief Description of the Paper

The paper by Kim (2014) implements a simple CNN with only one convolutional layer on various datasets like the Movie Review Data (MR) (Pang & Lee, 2005), TREC experimental data for question classification (Li & Roth, 2002), customer reviews (Hu & Liu, 2004), etc. Every word in the dataset is converted into a vector of a fixed dimension by either random initialization or word2vec trained on 100 billion words from Google News (Mikolov, Sutskever, Chen, Corrado, & Dean, 2013). The paper evaluates different variants of the model, as listed below:

a) CNN-rand: Vectorizing all words randomly, and updating the vectors during training
b) CNN-static: Vectorizing all words by publicly available pre-trained word2wec to 300 dimensions, and keeping the vectors static during training
c) CNN-non-static: Same as CNN-static, but changing the vectors during training according to specific tasks
d) CNN-multichannel: Vectorizing all words by word2vec to two sets of vectors, one of which is static and the other is non-static

CNN-Static is chosen as our baseline model.

Figure 1 (Kim) shows the architecture of the model, which is similar to the CNN architecture implemented by Collobert et al. (2011). For every sentence input, each of the words is converted to a k-dimensional vector. The vectors are concatenated together, which means a sentence of length n is represented by an n×k matrix. Then in the convolutional layer, a filter of size h×k is applied to generate new features. Thus, for each filter, the convolutional layer generates new n-k+1 dimension features. These features are then passed to a max-over-time pooling (Collobert et al.) layer. Multiple filters of different size are used by the model, and one feature is extracted through each filter. A fully connected softmax layer is then applied to all the extracted features, whose output is a distribution of probabilities.

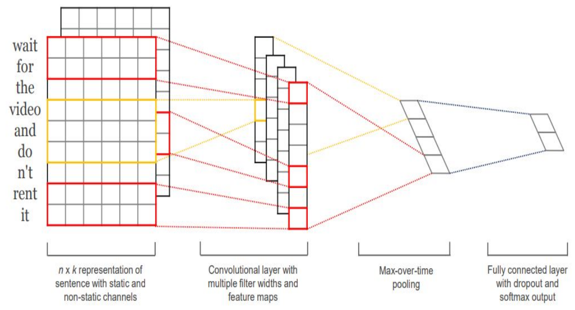
wait
for
the
video
and
do
n't
rent
it
n x k representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Figure 1: Model architecture proposed by Yoon Kim

III. Dataset

Though the paper did experiments on seven different datasets, our project focuses only on one dataset, which is the MR introduced by Pang and Lee (2005). Made up of 5331 positive and 5331 negative processed movie review sentences and/or snippets, MR was developed for sentiment polarity experiments. Note that since there is no standard split of training and validation set on this dataset, we followed the author and used 10-fold cross-validation to report the model performance with a 10% split for the validation set.

IV. Implementation of the Baseline

In this study, the CNN-static model with 81.0% prediction accuracy in the MR dataset is chosen as the target baseline. The model is reproduced by following a two-step process.

Firstly, the raw data (collection of sentences) is preprocessed by removing the punctuations. Individual words are then converted into vectors by utilizing the *word2vec* vectors from *Google News*. Due to the large dimensionality of the *word2vec* vectors, procedures such as stemming and lower casing of vocabularies are not required in this step. For the words that are missing in *word2vec*, we initialize their weight randomly. This is implemented by creating a 300-dimensional feature vector using normal distribution functions with a mean and standard deviation of -0.0036, and 0.118, respectively. These two values are calculated from the *word2vec* and are suitable to represent the vectors for missing words. Repeated missing words are assigned with the same word vector, for example, all the "a" in the database are converted into identical word vector even in different sentences. This proposed approach results in the creation of 1909 random vectors (1909 missing words) during the data pre-processing step. It is worth mentioning that the number of missing words reported in the paper are 2317. Although there is a small discrepancy for the missing word counts, it should have a negligible effect on our model considering the relatively large vocabulary size (18765).

Secondly, a CNN model is constructed following the abovementioned architecture by utilizing the negative log-likelihood as the loss function and the Adadelta update rule as the optimizer. The padding size is chosen to be one unit smaller than the size of filter windows. For example, a padding size of 2 will be used for a filter window size of 3 (*i.e.* kernel size 3 × 300). The model is trained by a mini-batch size of 50.

The result of the baseline model is evaluated by running the 10-fold cross-validations for 10 times and Table 1 shows the mean accuracy and standard deviation for each test. Average accuracy of 80.71% can be calculated from the results, which is close to the reported value of 81.0%. Although 10-fold cross-validation is implemented, we can still observe that the mean accuracy is fluctuating among all the tests. In this table, the accuracy ranges from 80.29% to 81.16%. Taking this into account, the 0.29% difference (81.00% - 80.71%) may be caused by the fluctuation of the predictions and we can conclude that the baseline model is successfully reproduced.

Table 1: The performance of the reproduced baseline model by using cross-validation.

Number of Tests	Mean Accuracy (%)	Standard Deviation (%)
1	80.75	0.81
2	80.29	0.66
3	80.72	0.85
4	80.88	1.14
5	80.79	0.72
6	80.57	1.19
7	81.00	0.62
8	80.34	0.69
9	80.59	0.63
10	81.16	0.80
Average	**80.71**	**0.81**

V. Improving the Baseline

We try to improve the performance of the baseline model by following three different routes, i.e. tuning the hyper-parameters, modifying the architecture and changing the word vector representation.

A. *Tuning the Hyper-parameters*

In this section, we conduct an extensive survey of hyper-parameters tuning.

a) Effect of the filter window size

We explore the effect of the size of the filter window. The number of filters (feature maps) is fixed to 100. We start with the baseline model with a filter window of (3,4,5). We observe (Figure 2) that even if we move away from the baseline filter window size, the performance remains almost constant.

We also try to implement a model with same filter window size multiple times i.e. (7,7,7) but it does not improve the performance of the model, as opposed to what Zhang and Wallace (2016) have suggested. This filter window size also results in the biggest standard deviation among all the tests.

Our experiments show that the best filter window size is (3,4,5) which is the default or (7,8,9) which gives similar accuracy but a smaller standard deviation. We will choose a filter window size from these two choices in combination with other hyper-parameters.

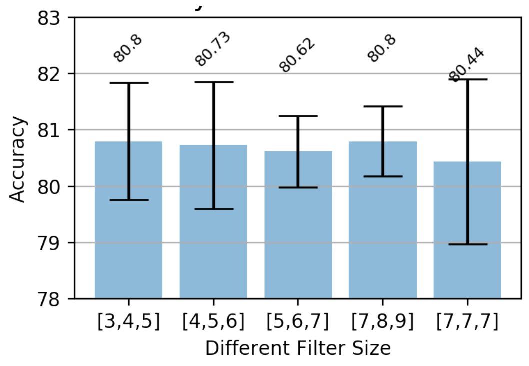

Figure 2: Accuracy for different filter window size

b) Effect of the number of feature maps

We try to experiment with the number of feature maps used in the convolutional layer of the model. The default number of 100 is experimented with and replaced by 10, 50, 200, 400, 600 and 1000.

We observed that increasing the number of filters beyond 100 does not help much in terms of the model's accuracy but hurts a little in terms of time taken to converge. As can be seen from Figure 3, accuracy drops by almost 1% as we increase the number of feature maps from 100 to 1000. Our analysis shows that the default number of feature maps gives the highest accuracy.

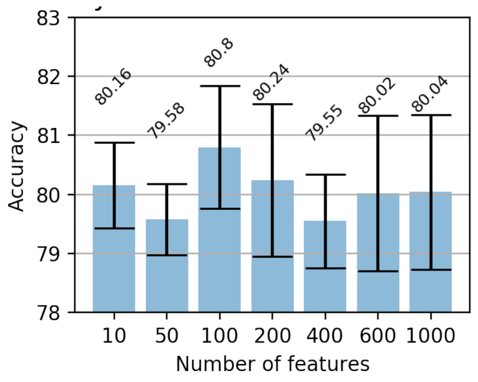

Figure 3: Accuracy for different number of feature maps

c) Effect of activation functions

We consider four different activation functions for the convolutional layers. Kim (2014) used Rectified Linear Units as the choice of the activation function. Hence, starting with ReLU we explored other activation functions as Sigmoid (Mass et al., 2013), tanh (hyperbolic tangent) and Softplus (Dugas et al., 2001). Results are presented in Figure 4 below, tested on the baseline model.

The best results are achieved using ReLU as the activation function, while sigmoid gives the worst performance. ReLU has the advantage of non-saturating form compared to sigmoid. Zhang et al. (2016) report that ReLU allows the SGD optimizer to converge at a faster pace compared to the other activation functions. As stated, sigmoid saturates very easily on both sides of the origin, the accuracy obtained is very small compared to the other three. On performing the experiments and observing the results shown in the table below, we will suggest keeping ReLU as the activation function. Not only does ReLU gives the best accuracy, but also the smallest standard deviation. ReLU is followed by tanh and then Softplus in terms of model accuracy. And the standard deviation follows the same pattern.

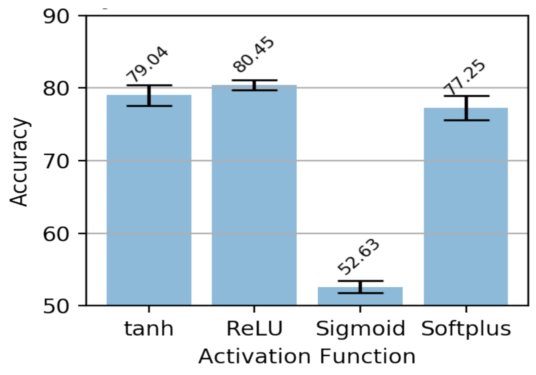

Figure 4: Accuracy for different activation functions

d) Effect of regularization

Two types of regularizations discussed in the paper are the dropout rate and the l2 norm constraint. Dropout is applied to the penultimate layer. We try the values of dropout ranging from 0 to 0.9. While experimenting with the dropout rate, the l2 norm constraint is fixed at 3, which is the baseline configuration. We observe that the default dropout rate of 0.5 gives the best accuracy and the smallest standard deviation. A dropout rate of 0.7 gives similar accuracy but with a slightly higher standard deviation.

Kim also applies l2 norm constraint to the penultimate layer of the model, which imposes a constraint on the weight vectors that parameterize the softmax function. A smaller l2 norm of 0.01 gives the highest accuracy (Figure 6).

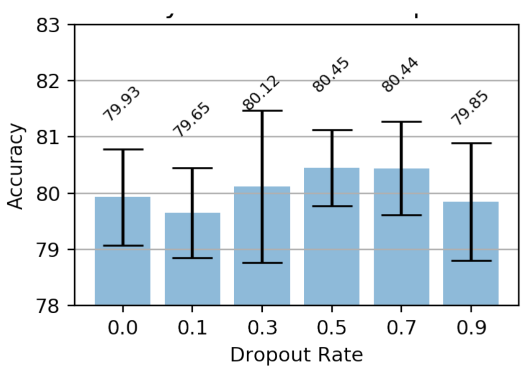

Figure 5: Accuracy for different dropout rates

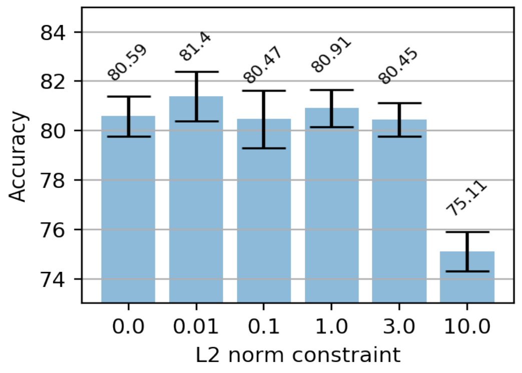

Figure 6: Accuracy for different l2-norm constraint

e) Effect of optimizers

We implement different optimization algorithms and fine-tune their hyper-parameters to explore the effects. Adadelta (Zeiler, 2012) update rule was used by Kim (2014). Figure 7 shows the influence of different learning rates (lr), and Figure 8 shows the influence of different rho (the parameter responsible for getting the average of squared gradients). Among all the parameters tested, the combination of lr = 0.2 and rho = 0.9 gives us the best performance, which is an accuracy of 80.97 with a standard deviation of 0.97.

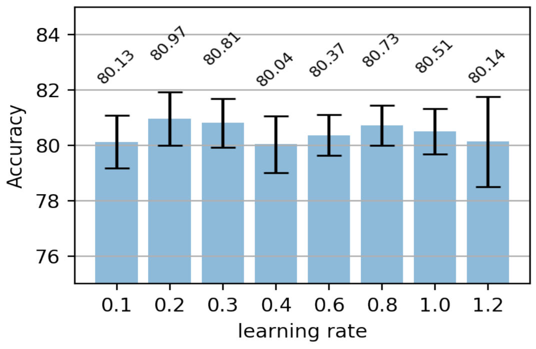

Figure 7: Accuracy of Adadelta for different learning rates

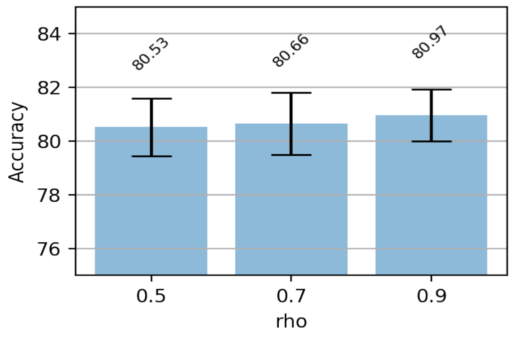

Figure 8: Accuracy of Adadelta for different rho with a learning rate of 0.2

Note that when using Adadelta, the learning is fast in the early stages but the loss in later epochs is alternating between several local extrema (Figure 9). This can be the reason for the fluctuations mentioned in the earlier section.

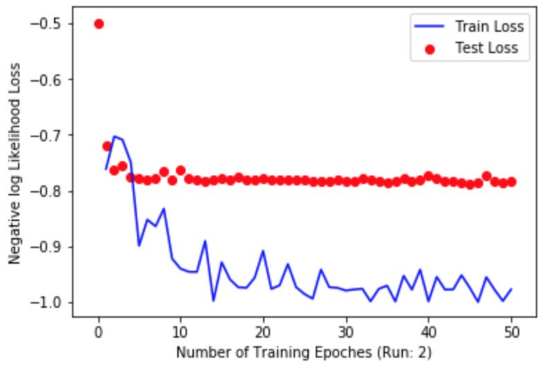

Figure 9: Training and test loss vs the number of epochs

We also try Adagrad (Duchi, Hazan, & Singer, 2011) update rule. The accuracy of different learning rates is given in Figure 10. This is consistent with the finding of the author that Adagrad and Adadelta gave us similar results.

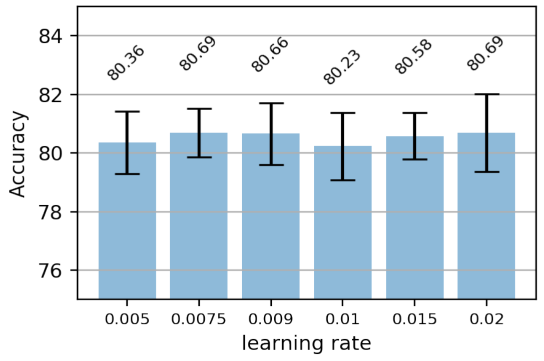

Figure 10: Accuracy of Adagrad for different learning rates

The Adam (Kingma & Ba, 2014) optimization algorithm is the de facto standard in deep learning. Fine-tuning its hyper-parameters improves the model performance. Figure 11 shows the influence of different learning rates (lr), and Figure 12 shows the influence of different betas (the parameter responsible for getting the averages of gradients and its square). Among all the parameters tested, the combination of lr = 0.0002 and betas = [0.8, 0.999] gives us the best performance, which is an accuracy of 81.24 with a standard deviation of 1.63.

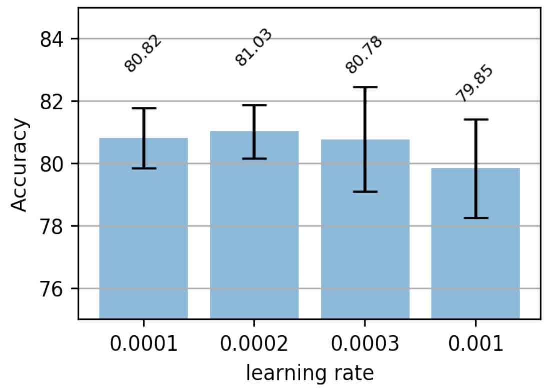

Figure 11: Accuracy of the optimizers Adam for different learning rates

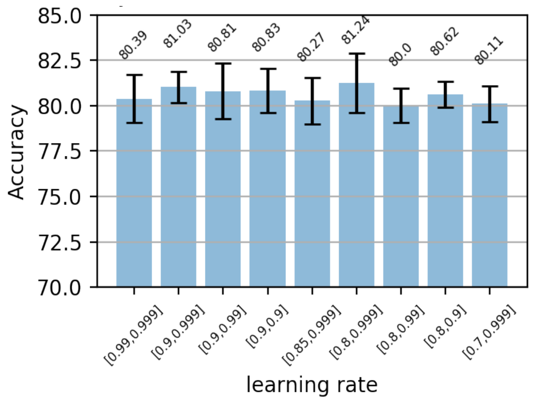

Figure 12: Accuracy of the optimizer Adam for different values of beta for a learning rate of 0.0002

Table 2: The best hyperparameters after tuning

Description	Values
Filter region size	[3,4,5]
Number of Filters	100
Activation function	ReLU
Dropout	0.5
l2 norm constraint	0.01
Optimizer	Adam (lr = 0.0002 and betas = [0.8, 0.999])

B. Architecture modification

Other than tuning the hyper-parameters, we also study the model by modifying its architecture. Compared to the baseline model which uses a filter window size of [3, 4, 5] (where each element translates to the use of 3-grams, 4-grams, and 5-grams respectively), we studied the performance of models by using only single or double filters. Our goal is to study the models with less complexity and evaluate their potential to beat the baseline. To mitigate the phenomenon of fluctuations in the predicted accuracy across each run (discussed in Section IV), all the results reported here are averaged by running the 10-fold cross-validation 4 times.

The result of using single N-gram(s) as the filter window size is presented in Figure 13. From the figure, we can observe that 4-grams has the highest accuracy of 80.43% among all trials, which is already close to the baseline accuracy of 80.71%. Except for the using of single 1-gram, all the other trials can achieve an accuracy exceeding 79.5% which is the highest accuracy reported in the article without using CNN (Dong et al. 2014). Although none of the attempts can beat the baseline model, it surely gives us an insight into the amount of information that different filter window sizes can extract from the dataset. Thus, for the following study on double filter window sizes, we mainly focus on the combination of filters having a higher accuracy (2 to 7-grams).

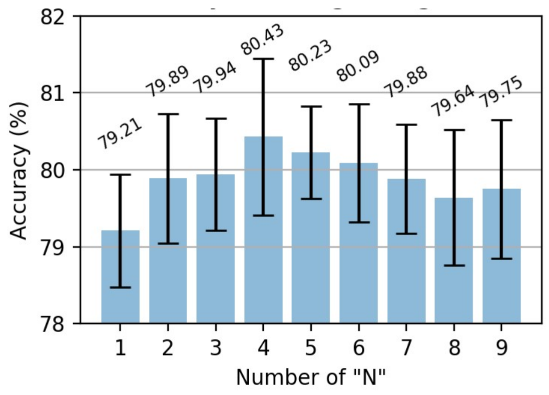

Figure 13: Accuracy for single N-gram(s)

The study of double filter window sizes is designed by using the linear combination of filters with sizes between 2 to 7. This results in a total of 15 different combinations, and their prediction accuracies are presented in Figure 14. From the figure, we can see that the model is doing surprisingly well when the second filter window is added (compared to the models with only one filter window). Seven out of fifteen models can obtain a mean accuracy exceeding 80.43% which is the highest value that a single filter can achieve. Among all the modified architectures, we can beat the baseline by using a combination of 2-grams and 6-grams filter windows which gives a prediction accuracy of 80.93% after running 10-fold cross-validation for 4 times. This result is rather surprising since it doesn't involve a 4-grams filter window, which gives the highest information extraction ability as seen in Figure 13. Nevertheless, the prediction involving the use of 4-grams filter window still works consistently well (except the example of 3, 4-grams).

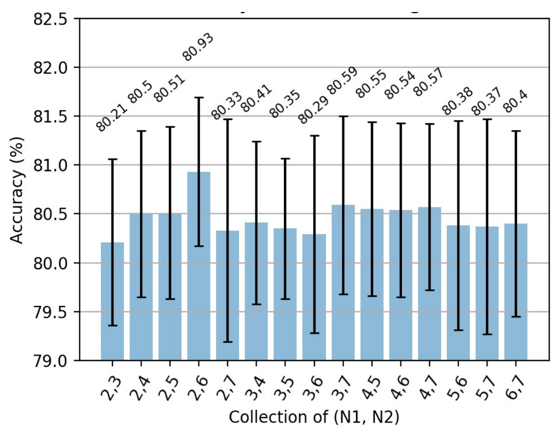

Figure 14: The prediction accuracy of the modified models using double N-grams filters.

Lastly, the combinations of multiple N-grams are also investigated in our research. The results of using 4 and 6 different filter windows are shown in Figure 15. For the study of using 4 different filter windows, we tried to combine the baseline model (3,4,5) with an extra filter window. Filter window size for 1, 2 and 6-grams are selected here. According to the discussion for single N-gram(s), we know that single 6-grams filter window extracts the most information among these three choices, while 1-gram has the least. We were anticipating a similar trend on adding these to the baseline model. Although none of the attempts improves the baseline, we can observe that the introduction of 1-gram filter window, which has the worst performance among the three when used individually, surprisingly gives the best result. Inspired by this discovery, we try to combine the original model with 7, 8 and 9-gram filter windows which also perform relatively poor when using them individually. From Figure 15, we can observe that the combination of these 6 filters gives us an accuracy of 80.73% which only slightly beats our baseline model, but it increases the computational time significantly. These observations reveal the fact that utilizing complex models (number of filters > 3) may not be an ideal option to tackle this dataset.

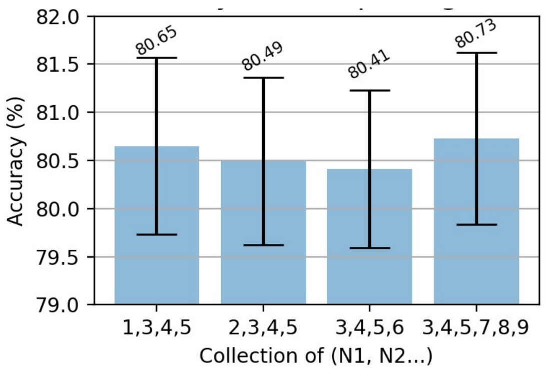

Figure 15: The prediction accuracy of the modified models using 4 and 5 filters

C. Changing the word vector representation

One of the most popular representations of document vocabulary is 'Word Embedding'. Word Embeddings are omnipresent in Natural Language Processing (NLP). Several researchers and academicians have developed and shared word embeddings trained on a large dataset (Collobert et al., 2011; Mikolov et al., 2013; Pennington et al., 2014). And these embeddings have been used effectively for many classification tasks (Turian et al., 2010; Socher et al., 2011; Kim, 2014; Bansal et al., 2014; Tai et al., 2015). Word embedding not only captures the context of the word in a document but also captures the semantic and syntactic similarity and the relation with other words. In simple terms, word embedding is the representation of a word using a vector of fixed dimension. For example, the *word2vec* used in our model is obtained from *Google News*.

There are several other word representations that can be used in place of Google News. One such word representation is *GloVe* (Jeffrey Pennington 2014). *GloVe* combines elements from the two main word embedding models, i.e. global matrix factorization and local context window methods. The *GloVe* word representation we use is trained on 6 billion words with each word represented as a vector of dimension 200. We go even a step further by concatenating *Google News* vector and the *GloVe* word representations. The results can be seen in Figure 16.

Although using *GloVe* representation only does not improve the performance of the model for both types of embeddings (static and non-static) but combining *Google News* vector and *GloVe* representation improves the performance of the model slightly. Concatenating 300-dimensional vectors from *Google News* and 200-dimensional vector from *GloVe* representation gives us a 500-dimensional vector which can be used as an input to the model. This implementation gives us an accuracy slightly higher than the one reported by the paper for non-static embedding (Figure 16). One future work can be replacing the word representation with another model and try to achieve better accuracy because the new model may have more words that are present in the movie reviews.

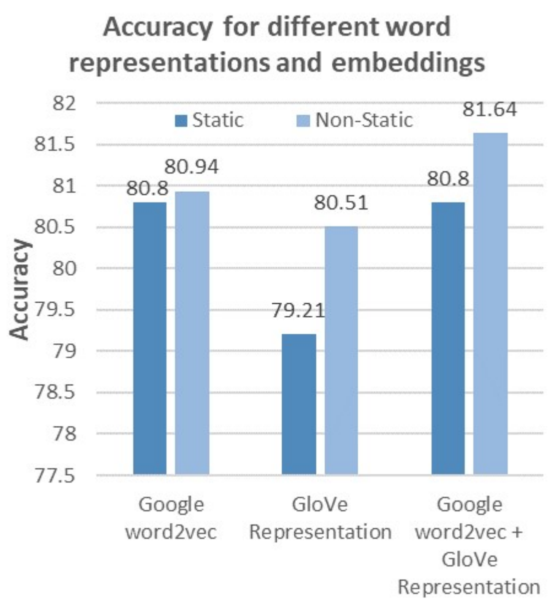

Figure 16: Accuracy for different types of embeddings for different word representations

DISCUSSION AND CONCLUSION

In our experiment, because of randomness, the accuracy can vary by more than 1% among different runs using the same set of hyper-parameters. As Kim (2014) kept the "CV-fold assignment, initialization of unknown word vectors, initialization of CNN parameters" constant to reduce randomness, it is possible that the accuracy reported by Kim is higher than our results due to the choice of these random values.

In general, the simple CNN model used by Kim works quite well for the MR dataset. Our project reproduces the CNN-static baseline model successfully with an understandable difference. Fine-tuning the hyper-parameters does not give us a stable improvement on accuracy. However, if considering the CNN-non-static model, adding the Glove word representation can give us a more than 0.1 stable improvement on accuracy compared to the CNN-non-static benchmark (81.5).

In our project, all the analysis of improving the baseline model is conducted ceteris paribus. Further study may be done by changing different hyper-parameters, architecture, and word vector representation together. Because if we modify some variables separately, it may not give us much improvement on the accuracy. Yet, it may improve the performance of the model a lot when changing them together.

Statement of contributions: Our group had several meetings, and we chose the paper, reproduced the baseline model, fine-tuned the hyper-parameters and wrote the reports together. Overall, everyone contributed equally to this project.

REFERENCES

[1] Bansal M., Gimpel M. and Livescu M. (2014) Tailoring Continuous Word Representations for Dependency Parsing

[2] Collobert R, Weston J., Bottou L., Karlen M., Kavukcuoglu K, Kuksa P. (2011). Natural Language Processing (almost) from Scratch

[3] Dong, L., Wei, F., Liu, S., Zhou, M., & Xu, K. (2015). A statistical parsing framework for sentiment classification. Computational Linguistics, 41(2), 293-336.

[4] Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12(Jul), 2121-2159.

[5] Dugas C., Bengio Y. , Belisle F., Nadeau C., Garcia R. (2001) Incorporating Second-Order Functional Knowledge for Better Option Pricing

[6] Hu, M., & Liu, B. (2004, August). Mining and summarizing customer reviews. In Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 168-177). ACM.

[7] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[8] Li, X., & Roth, D. (2002, August). Learning question classifiers. In Proceedings of the 19th international conference on Computational linguistics-Volume 1 (pp. 1-7). Association for Computational Linguistics.

[9] Maas A. L, Hannun A. Y., Ng A. Y. (2013) Rectifier nonlinearities improve neural network acoustic models

[10] Mikolov T, Sutskever I, Chen K, Corrado GS, Dean J (2013) Distributed representations of words and phrases and their compositionality

[11] Pang, B., & Lee, L. (2005, June). Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In Proceedings of the 43rd annual meeting on association for computational linguistics (pp. 115-124). Association for Computational Linguistics.

[12] Pennington J., Socher R., Manning C.D. (2014) GloVe: Global Vectors for Word Representation

[13] Socher R., Perelygin A., Wu J.Y. , Chuang J., Manning C. D. , Ng A. Y. and Potts C. (2011) Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank

[14] Tai K. S., Socher R., Manning C D. (2015) Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks

[15] Tieleman, T., & Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning, 4(2), 26-31.

[16] Turian J., Ratinov L. , Bengio Y. (2010) Word representations: a simple and general method for semi-supervised learning

[17] Zeiler, M. D. (2012). ADADELTA: an adaptive learning rate method. arXiv preprint arXiv:1212.5701.

[18] Zhang Y., Wallace B. (2016) A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification