COMP 551 - Mini-Project 4 - Group 2

Alex Hale 260672475
Erick Zhao 260687719
Veronica Nasseem 260654054

Abstract

This paper analyzes the work of Kim [2014] on the use of convolutional neural networks (CNNs) for the task of sentence classification. The main goals are to reproduce the baseline model (a CNN operating on word vectors with randomly initialized weights), to tune the baseline model's hyperparameters and use a grid search technique to improve its performance, and to attempt to beat the baseline model's performance using simple models with extensively-tuned hyperparameters. with a simple model. While the baseline model's performance was not surpassed by these implementations, comparable results were achieved with the tuned CNN-rand model and the long short-term memory model.

1 Introduction

Machine learning researchers often achieve high performance on tasks by spending a lot of time and resources finely tuning complicated models. While this approach has the highest rate of success in achieving the best possible results, equivalent or superior results can sometimes be achieved by finely tuning simple models, which requires less time and resources than creating and tuning a complicated model. The goal of this project is to reproduce prior results on the task of using convolutional neural networks (CNNs) for sentence classification by Kim [2014]. The original paper shows that on a data set of features extracted with pre-trained word vectors, simple CNNs (without extensive hyperparameter tuning) perform competitively with state-of-the-art techniques for sentence classification (e.g. recursive autoencoders, matrix-vector recursive neural networks, combinatorial category autoencoders, etc.). To demonstrate these findings, the original paper ran experiments with four variants of the simple CNN model on seven different data sets.

The tasks completed in this paper are two-fold. First, the performance of the original paper's baseline model (CNN with random vector weights) is reproduced. That baseline is then further improved with extensive hyperparameter tuning. Second, two additional models which are even simpler than the CNN from the original paper are implemented - namely, a long short-term memory (LSTM) model and a Naive Bayes model, with the goal of surpassing the original paper's performance. These experiments are conducted on the Movie Review data set from Pang and Lee [2005], which is the first of the seven data sets from the original paper.

2 Related Work

2.1 Word Vectors

There are two general ways of encoding a corpus of text data for use in statistical classifiers: one-hot encoding and dense word vectors. One-hot encoding represents each word as a sparse (i.e. full of zeros) vector that is the length of the entire vocabulary. A single 1 digit in the sparse vector represents the identity of the word, with each unique word in the vocabulary having a designated index in the vector. Conversely, word vectors capture more meaning from the data by considering the sentence at large, rather than the individual word. Word vectors encode text into dense vectors, unlike the sparse vectors used by one-hot encoding [Goldberg, 2017].

Due to their dense vector format, word vectors have the advantage of being computationally less expensive, which is useful when training computation-heavy deep learning algorithms. More importantly, they allow for better generalization, because the feature space is smaller and each feature yields more information [Goldberg, 2017].

Kim [2014]'s CNN model makes heavy use of the popular `Word2Vec` word embeddings, which were trained by Mikolov et al. [2013] on a set of 100 billion words gathered from Google News.

2.2 Comparison with Other Models

The original paper by Kim [2014] concluded that a CNN with one convolution layer, trained with pre-trained word vectors, yielded better results on sentence classification tasks than other state-of-the-art techniques. In the five years since its publication, many papers have aimed to improve on its results. While this paper's implementation will only makes use of the Movie Review (MR) data set from Pang and Lee [2005], results from the other data sets mentioned in Kim's original are displayed in Table 1 for comparison.

Model	MR	TREC	SUBJ	CR	SST-1	SST-2	MPQA
CNN-static [Kim, 2014]	81.0	92.8	93.0	84.7	45.5	86.8	89.6
CNN-non-static [Kim, 2014]	81.5	93.6	93.4	84.3	48.0	87.2	89.5
CNN-multichannel [Kim, 2014]	81.1	92.2	93.2	85.0	47.4	88.1	89.4
BLSTM-2DCNN [Zhou et al., 2016]	82.3	**96.1**	94.0	-	**52.4**	**89.5**	-
AdaSent [Zhao et al., 2015]	**83.1**	92.4	**95.5**	**86.3**	-	-	**93.3**
AGRU [Kumar and Rastogi, 2019]	82.2	-	93.5	84.6	46.7	84.0	89.7
ABLSTM [Kumar and Rastogi, 2019]	81.9	-	92.9	83.2	45.9	84.0	89.4

Table 1: Subset of results from Kim [2014]. Column titles are data sets. Values represent percentage accuracy.

In Table 1, four different approaches which have generated good results when compared to Kim [2014] are highlighted:

- Zhou et al. [2016] used a bidirectional long short-term memory (LSTM) network with a 2-dimensional CNN to extract more meaning from the data.
- Zhao et al. [2015] created an adaptive hierarchical model of words, phrases, and sentences.
- Kumar and Rastogi [2019] focused on attention-based mechanisms in deep learning. Two of their model variations are noteworthy: attentional Gated Recurrent Units (AGRU) and an attentional bi-directional LSTM.

3 Data Set and Setup

While Kim [2014] tests their CNN models on 7 different data sets, this paper limits the scope to a single data set, because it reduces the computation time required to develop the models and tune their hyperparameters. The Movie Review data set from Pang and Lee [2005] is formed of 10662 one-sentence snippets from movie reviews written on Rotten Tomatoes, a review-aggregation website. Examples are labelled as either positive or negative, and the set contains an equal amount of examples from each label.

In their paper, Kim [2014] performs minimal pre-processing of the data, opting only to remove punctuation and perform tokenization before manually creating dense vectors as input for the CNN. This paper follows suit.

As in Kim [2014], this paper transforms the data into word vectors. Creating these word embeddings, captures the context of the sentence around a word in addition to the identity of the word itself. To prepare the data for the CNN-rand approach, the word vectors are initialized with random weights, then the first layer of the model (Keras' embedding layer) learns an embedding for each word in the set. For the basic model implementations, an additional basic one-hot encoding word vectorization is used as an additional form of hyperparameter tuning.

4 Proposed Approach

4.1 CNN-rand Reproduction

This task was to reproduce Kim [2014]'s CNN-rand model. The following architectural details were identified from their description of the model:

- An embedding layer with randomly initialized word vectors. Vectors are set to dimension 300 in their original model to match the dimension of the `Word2Vec` embeddings.
- At the next step, there are 3 two-dimensional convolutional layers that each operate on a window of varying size on multiple word vectors. (From the author's public GitHub code, the window sizes were identified as 3, 4, and 5. This was not mentioned in the original paper.) These convolutional layers are also regularized using L2 weight regularization.
- Each convolutional layer feeds into a two-dimensional max pooling layer.
- The parallel pooling layers are concatenated into a fully-connected layer that has dropout and a softmax output.

- The optimizer governing the learning process was the AdaDelta function, first introduced by Zeiler [2012].

This paper's implementation of the model used Keras with a TensorFlow backend. The built-in `Conv2D`, `Concatenate`, `Dense`, `Flatten`, `Reshape`, `Dropout`, and `MaxPool2D` layers were used. The architecture was similar to the description from Kim [2014], with a few changed details:

- The model was unable to train properly with a softmax activation on the output layer. A sigmoid activation function was used instead.

- Keras' built-in function was used to reduce the learning rate upon reaching a plateau in the validation loss.

With this implemented, there were numerous parameters available to tune, including learning rate (LR), LR reduction factor, number of epochs, batch size, filter window sizes, dropout rates, embedding layer dimensions, and more. A grid search was run using SciKit-Learn's built-in `GridSearchCV` to find the ideal set of hyperparameters. Varying the value for each of these hyperparameter value was computationally expensive, so without access to dedicated computational resources, only a small number of hyperparameter settings were tested (Table 2).

Hyperparameter	Values
Dropout rate	[0.5,0.9]
Learning rate	[0.1,1]
Epochs	[25,50]

Table 2: Hyperparameter combinations used during grid search

4.2 Alternative Model Implementations

The next task was to achieve similar performance to the results in Kim [2014] using models that are even simpler than their basic CNN. Such performance can be achieved by performing extensive hyperparameter tuning on the simple model, setting the hyperparameters to specific individual values for the data set under test. Each model is tested on two feature sets, both extracted from the Movie Review data set. The first feature extraction is a one-hot encoding of the words in each review, while the second approach is the more complex word-to-vector implementation outlined in section 3 (Data Set and Setup) above.

4.2.1 LSTM Model

The basic LSTM is a sequential neural network constructed in Keras using three layers. The first layer is an embedding layer, taking the encoded data set and converting it to vector embeddings that are compatible with the following layer. The LSTM layer operates on the output of the embedding layer with a parameterized number of memory units, then feeds its output to the 'Dense' layer to make a binary classification prediction using a sigmoid activation function. Assuming the feature extraction parameters are held constant, there are three variable hyperparameters in this model which can be tuned: the length of the embedding vector, the number of memory units in the LSTM, and the number of epochs to run.

4.2.2 Naive Bayes Model

The basic Naive Bayes model is SciKit-Learn's Multinomial Naive Bayes implementation (Pedregosa et al. [2011]). Multinomial Naive Bayes was chosen over the other Naive Bayes variants because it is listed as one of the competing models in Kim [2014]. Making the same feature extraction parameter assumption as was made for the LSTM, the variable hyperparameters for the Naive Bayes model are the amount of additive smoothing and whether or not to learn prior class probabilities.

5 Discussion of results

5.1 CNN-rand reproduction

The Keras CNN-rand implementation failed to reproduce the original paper's **76.1%** accuracy. The grid search found that a dropout rate of 0.5, 25 epochs, and a learning rate of 0.1 were the optimal hyperparameter settings, but their performance on the validation set was only **73%** accuracy.

class	precision	recall	f1-score	support
negative	0.70	0.78	0.73	1033
positive	0.76	0.69	0.72	1100
average	0.73	0.73	0.73	2133

Table 3: Results on validation set.

One hypothesis for the cause of these subpar results is that the limited grid search performed in this paper failed to yield more optimal hyperparameter settings that used in Kim [2014]. Another possibility is that the structure of the model was not interpreted properly from Kim [2014], or that there were some implementation details lost in translation between their Python2/Theano implementation and this Python3/Keras implementation.

5.2 Alternative model implementations

Hyperparameters for both models and both extracted feature sets were tuned by trial-and-error, iterating over a list of hyperparameter options to determine the performance for each. The results are reported below.

5.2.1 LSTM model

The basic LSTM was trained on both feature sets with embedding vector length ranging between 64 and 256, LSTM memory units ranging from 50 to 150, and epochs ranging from 1 to 15. When using the one-hot encoded features, the LSTM's best performance was 74.96%, achieved using 75 LSTM memory units, a vocabulary length of 128, and training for 3 epochs. When using the word-to-vector encoded features, the LSTM's best performance was 76.10%, achieved using 50 LSTM memory units, a vocabulary length of 64, and training for 10 epochs. The full results are charted in Figures 1 and 2, with peak accuracy indicated by a red dot.

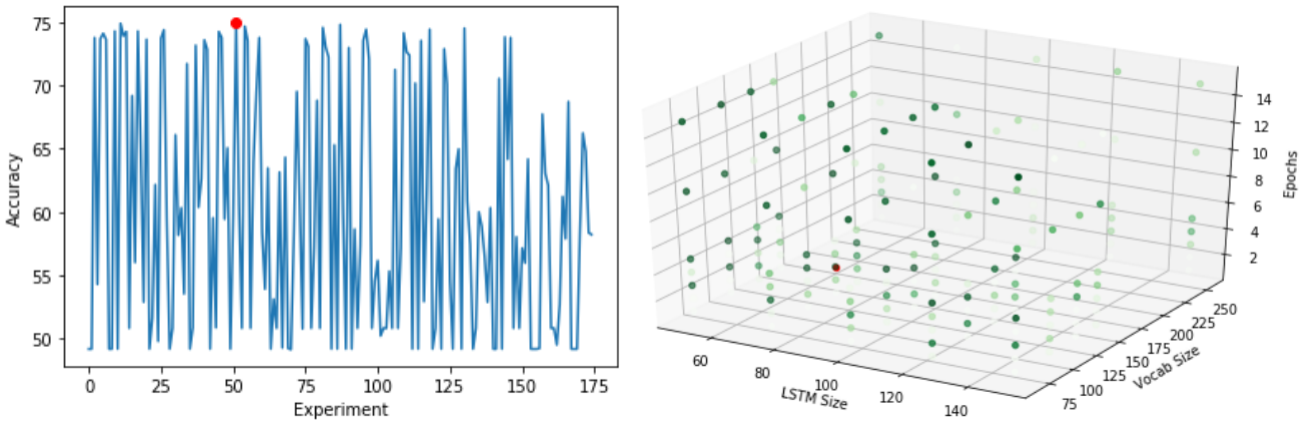

Figure 1: Hyperparameter tuning results for one-hot encoded data. Darker green dots indicate better accuracy; red dot indicates best accuracy.

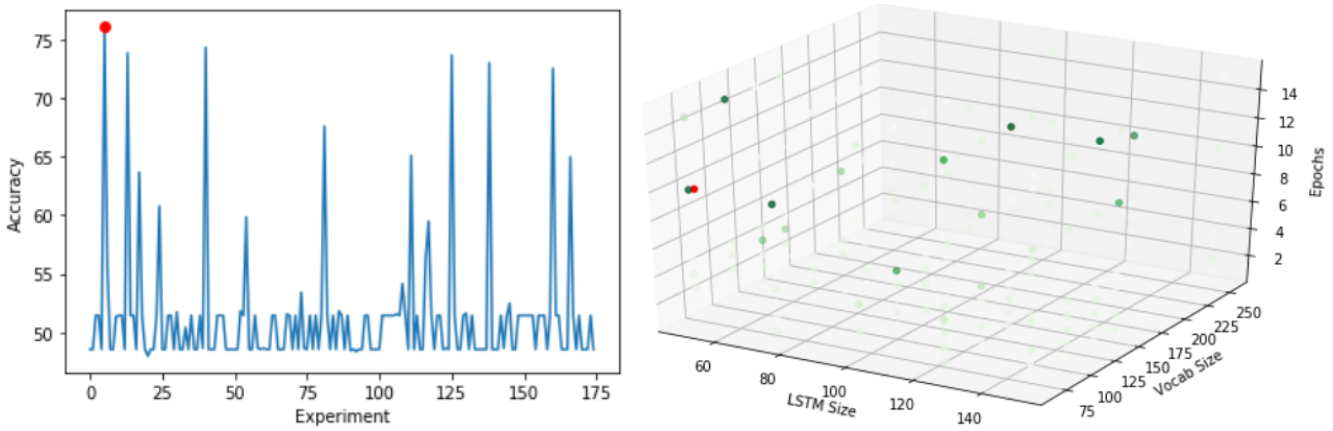

Figure 2: Hyperparameter tuning results for word-to-vector encoded data. Darker green dots indicate better accuracy; red dot indicates best accuracy.

These accuracy rates approach, but don't quite match those of the models in the original paper. It is expected that these poor results are due in part to the pre-processing of the data, rather than the hyperparameters of the models themselves. The one-hot method encodes much less information than the word-to-vector encoding method, so it is logical that it achieves lower performance than the models in Kim [2014], which used word-to-vector encoded features. It is expected that the word-to-vector feature extraction performed in this implementation is not fully compatible with the LSTM model, yielding lower-than-expected results. An additional comparison between the feature extraction techniques shows that the one-hot encoded features achieved a roughly equal number of high accuracy scores to low accuracy scores, but a lower peak accuracy, while the word-to-vector encoded features achieved many low accuracies and few high accuracies, but a higher peak accuracy.

5.2.2 Naive Bayes model

Multinomial Naive Bayes was unable to fit the data, achieving a constant near-random performance of 50.16% and 52.47% accuracy (on the one-hot-encoded and word-to-vector feature sets, respectively), regardless of the hyperparameter setting. This performance result remained constant for additive smoothing between 1e-10 and 100, both with and without learning prior class probabilities. These poor results indicate that the way in which features were extracted from the data set was not compatible as input to the Naive Bayes model, because the model would achieve performance better than random if any meaningful features were used.

6 Conclusion

This paper attempted to reproduce the work of Kim [2014] by replicating their baseline CNN architecture in Keras. The resulting classifier was functional, but did not match the performance of any of the CNN's from the original paper. Improved accuracy, but still not equal to the original paper's performance, was achieved using an LSTM. The Naive Bayes implementation was unsuccessful, obtaining near-random prediction results.

Further research could involve performing a more extensive grid search to uncover better hyperparameters for the CNN implementation, tuning other types of simple models, performing additional data-preprocessing, or ensuring that the word-to-vector encoding is compatible with the model under test.

7 Statement of contribution

- Alex Hale: Implementation and analysis of LSTM and Naive Bayes models.
- Erick Zhao: Reproducing the CNN-rand architecture, data processing, background research on the paper.
- Veronica Nasseem: Hyperparameter tuning in CNN, background research.

References

Y. Goldberg. Neural network methods for natural language processing. *Synthesis Lectures on Human Language Technologies*, 10(1):1–309, 2017. doi: 10.2200/S00762ED1V01Y201703HLT037. URL https://doi.org/10.2200/S00762ED1V01Y201703HLT037.

Y. Kim. Convolutional neural networks for sentence classification, 2014.

A. Kumar and R. Rastogi. Attentional recurrent neural networks for sentence classification. In D. Deb, V. E. Balas, and R. Dey, editors, *Innovations in Infrastructure*, pages 549–559, Singapore, 2019. Springer Singapore. ISBN 978-981-13-1966-2.

T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality, 2013.

B. Pang and L. Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of ACL*, pages 115–124, 2005.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

M. D. Zeiler. Adadelta: An adaptive learning rate method, 2012.

H. Zhao, Z. Lu, and P. Poupart. Self-adaptive hierarchical sentence model, 2015.

P. Zhou, Z. Qi, S. Zheng, J. Xu, H. Bao, and B. Xu. Text classification improved by integrating bidirectional lstm with two-dimensional max pooling, 2016.