Applying BERT to IMDB Movie Reviews

Eugene Tang
eugenet@berkeley.edu

Jesse Calvillo
jlc@berkeley.edu

Abstract

BERT embeddings have been shown to produce state-of-the-art results in many NLP tasks. For this project, we seek to apply BERT embeddings on the IMDB sentiment analysis task, where we need to classify IMDB reviews as positive or negative sentiment. Our best model has an accuracy of 89.21% compared to a baseline of 50.00%. While we did not achieve state-of-the-art results in this task, our experiments have found some promising avenues through which to further improve our model, including embedding more tokens of a review, using the front and back of a review if we are limited in the number of tokens we can embed, and improving our architecture.

1 Introduction

Sentiment analysis is a classic task in NLP that has a wide range of applications. Classically, word dictionaries such as LIWC (Tausczik and Pennebaker, 2010) have been used to determine sentiment, counting the number of positive and negative words in a sentence. However, simply counting words has its limitations, such as not being able to distinguish that ”not good” actually has a negative sentiment. Recently, a plethora of embedding techniques have arisen with promise to resolve some of these issues. In particular, the BERT embedding technique has produced some state-of-the-art results in many NLP tasks (Devlin et al., 2018). In our project, we explore the potential of BERT embeddings in sentiment analysis by applying them to classify IMDB movie reviews from the IMDB movie review dataset (Maas et al., 2011) to see if we can get close to state-of-the-art results. Namely, we create a BERT embedding-based sentiment classifier and explore the effect of different parameters, such as the number of dimensions used in the BERT embedding on the accuracy of the resulting model.

2 Background

The IMDB movie review dataset was first introduced by Maas et al. in 2011 (Maas et al., 2011). It contains 50,000 IMDB movie reviews. 25,000 are used for training while the other 25,000 are set aside for testing. Of the training and testing groups, 50% of the review are positive ($\geq$ 7 out of 10 stars), and 50% of the reviews are negative ($\leq$ 4 out of 10 stars). All ”neutral” reviews were removed.

To address this task, Maas et al. train embeddings using both an unsupervised semantic-based technique similar to LDA and a supervised technique to encode the sentiment of each word in its embedding (Maas et al., 2011). Afterwards, they feed the embeddings as a bag-of-words into a linear SVM to make their predictions. Through this technique, they are able to obtain an 88.89% accuracy.

Since then, various neural network techniques have been used to improve the classification performance on the dataset. Techniques include adding context vectors in addition to the word vectors as features (McCann et al., 2017), adversarial learning (Miyato et al., 2016), and LSTMs to embed a series of text directly (Johnson and Zhang, 2016). The current state-of-the-art is from the Universal Language Model Fine-tuning for Text Classification (ULMFiT) (Howard and Ruder, 2018). It achieves a 95.4% accuracy by using a language model trained with transfer learning and allowing the language model to be updated, or ”fine-tuned,” to address the particular task at hand.

BERT is a language representation model that has been producing state-of-the-art results across many NLP tasks (Devlin et al., 2018). The main innovation of BERT is that it conditions on both left and right contexts when deriving its representation at each layer. When the paper was published, it achieved 11 state-of-the-art results on standardized NLP tasks, leading to others successfully using BERT to achieve state-of-the-art results on additional tasks. For this reason, in this project we hope to see if we can use BERT embeddings to achieve strong performance on the IMDB classification task.

3 Methodology

3.1 Architecture

Because BERT was created as a fine-tuning representation model, one can take pre-existing BERT models and update the parameters for the task at hand (Devlin et al., 2018). Specifically, the authors fine-tuned BERT for future tasks by adding a classification layer on top of the BERT embeddings and then tuned the classification layer as well as some BERT parameters (Devlin et al., 2018).

Due to compute constraints (both time and space), we perform a variant where we compute the embeddings for every review with a pretrained BERT model using `bert-as-a-service`.[1] We use the embeddings as input to a neural network with one hidden layer trained to classify the reviews as positive or negative.[2] The disadvantages of this technique is that we no longer are able to fine-tune the parameters of BERT. However, this technique does allow us to perform many experiments with our precomputed BERT embeddings.

Throughout our experiments, we evaluate the effect of various parameters, namely:

- BERT model type: There are four English BERT models. These models are the cross-section of whether the casing in the documents was preserved (cased v. uncased) as well as the model size (small v. large). The small model has 12 layers, 768-dimensional embeddings with 12 self-attention heads. The large model has 24 layers, 1024-dimensional embeddings with 24 self-attention heads (Devlin et al., 2018).
- Embedded review length: Due to compute constraints, it is a lot faster to compute a subset of a review rather than the entire review. We test embedding the first 100 and 200 tokens of a review.
- Embedded review parts: Due to the restrained number of tokens we could embed, we experiment with embedding different parts of a review.
- Number of hidden units in the ”fine-tuning” layer: We test varying numbers of hidden units in the neural network we add on top.
- Number of training epochs: We test the effect of the number of training epochs
- Activation function: We test varying activation functions, namely the ReLU, Softmax, and tanh functions.
- Dropout level: We test the effect of the dropout level of the network.
- Optimizer: We test two optimizers, the AdaGrad optimizer (Duchi et al., 2011), which has been found to work well when training neural networks, and the Adam weighted optimizer, which is what BERT was trained using (Devlin et al., 2018).
- Number of Training Examples: We examine how sensitive the model is to the number of training examples it receives.
- Pooling strategy: BERT applies pooling at the second to last layer (because the original architecture target is focused on next sentence prediction and the final layer is specific to that task). Each layer in the model captures different information: we examine the performance differences between no pooling, average pooling, and max pooling, as well as first/last (special ”fine-tuning” token) pooling.
- Learning rate: The AdaGrad optimizer’s learning rates which are too high or two low will result in sub-optimal model weights. We perform a manual grid search to determine an appropriate learning rate range.

After selecting our topic and beginning experiments, we found that Google actually has an implementation of fine-tuning BERT embeddings for

[1] `https://github.com/hanxiao/bert-as-service`

[2] `https://www.tensorflow.org/api_docs/python/tf/estimator/DNNClassifier`

IMDB movie reviews.[3] This test was run on the uncased, small model. Due to RAM compute limitations, we were unable to extend our tests above to run on Google's architecture. Given more time, we might see additional improvements from performing our above tests on the Google architecture.

3.2 Data and Evaluation

In all of our tests, we split the original 25,000 training dataset into 20,000 training examples and 5,000 dev examples. To measure the quality of our models, we train them on the 20,000 training examples and evaluate their accuracy on the 5,000 dev samples. After we have fixed a set of parameters, we will train it on the 25,000 training dataset and evaluate its accuracy on the 25,000 testing dataset. We will use accuracy (how many reviews it classified correctly) as our metric on which we compare our model to other paper's. This is also the metric all the other papers use for comparison.

4 Results and Discussion

4.1 EDA

Dataset	Unique Movies in Positive Reviews	Unique Movies in Negative Reviews
Train	1390	2957
Test	1351	3016

Table 1: Number of unique movies in each data sample.

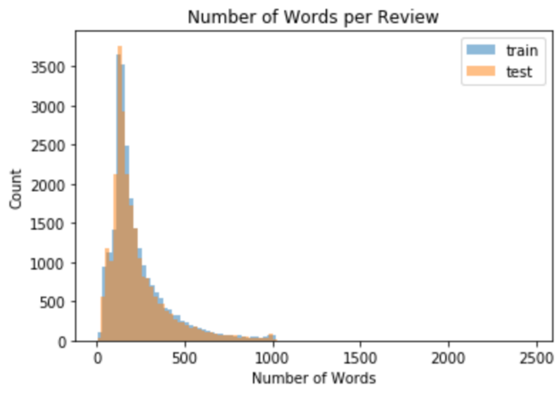

Figure 1: Number of Words per Review

First, we present some metrics on the dataset performed through EDA. The dataset contains 25,000 training and 25,000 testing examples, both containing a 50/50 split of positive and negative reviews. There are more unique movies in the negative review portion of the dataset than the positive review portion, but both the test and train dataset have similar numbers of unique movies as shown in Table 1. The reviews have a median of 174 tokens per review and a mean of 233 tokens per review. The shortest review had 10 tokens while the longest had 2470. That being said, Figure 1 shows the distribution of the number of tokens in the training and testing datasets are again fairly similar.

Looking at the words in each review, we find that words like "excellent", "perfect", and "loved" tend to appear much more often in positive reviews (3-4x more occurrences in positive reviews) while words like "supposed", "bad", and "poor" appear much more often in negative reviews (3-4x more occurrences in negative reviews). These make sense and do seem to indicate that the reviews are reasonably split into the positive and negative categories.

4.2 Effect of Parameters

Here we present the effects of various parameters on the accuracy of the resulting model. Table 2 shows our default parameters on all the tests unless stated otherwise.

Parameter	Value
BERT model type	uncased_L24_H1024_A16
Embedded review length	200
Number of hidden units	1024
Number of training epochs	30
Activation function	ReLU
Dropout level	0.1
Optimizer	0.1
Pooling strategy	Mean Pooling
Learning rate	0.003

Table 2: Default parameters.

[3]https://colab.research.google.com/github/google-research/bert/blob/master/predicting_movie_reviews_with_bert_on_tf_hub.ipynb

4.2.1 Bert Model Type

Model Type	Accuracy
Small, Uncased	0.8198
Small, Cased	0.8094
Large, Uncased	0.8512
Large, Cased	0.8342

Table 3: Effect of BERT model type, only embedding the first 100 tokens of a review.

To expedite the speed of the tests, the results here are based on the first 100 tokens of a review. Table 7 the uncased models generally performed better than the cased models. This might make sense because making the sentence uncased would allow the model to be more generalizable rather than overfitting to how certain capitalized words are used.

In addition, we found that the large model performed better than the small one. This is expected because: it contains more embedding dimensions and is a deeper network, so it is able to learn ”more” and represent more in its embeddings.

4.2.2 Number of Embedded Tokens

Number of Embedded Tokens	Accuracy
100	0.8512
200	0.8930

Table 4: Effect of number of embedded tokens.

Table 4 shows that the number of embedded tokens is one of the largest factors affecting the performance of our models. The more tokens of a review we embedded, the better the model did. This would make sense because embedding more tokens gives the model more data to decide from.

4.2.3 Parts of a Review Embedded

One thing we noticed is that the front and back of long reviews generally contain good summaries of a reviewer's sentiment toward the movie. For example, one sample review reads in the beginning ”Very sweet pilot. The show reeks of Tim Burton's better films” and ends with ”Two underrated character actresses that never fail to bring it with their performances” while the middle contains many details on the movie such as ”Every time I see her on the screen I see Zooey”. Given that we were limiting the number of tokens in a review we would embed, one hypothesis is that we would have a better embedding of the review if we took half our tokens from the beginning of a review and half from the end.

A test of this with the uncased small BERT model embedding only 100 tokens did prove promising, improving the results from 81.98% to 85.78%. However, it did not seem to extend as well when we embedded the first 100 and last 100 tokens of a review with the uncased large BERT model, only increasing performance from 89.30% to 89.34%. Preliminary tests with different parameters showed that the last 50 words of a review might even be more predictive than the first 50 (75.46% v. 76.20%), and this would be an area we want to explore further if given more time.

4.2.4 Number of Hidden Units

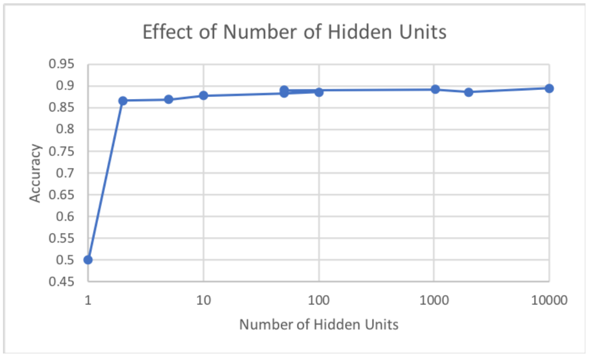

Figure 2: Effect of the Number of Hidden Units

Figure 2 shows that the more hidden units we used, the better the model performed. Performance stopped improving after around 1024 hidden units.

In addition, we performed a test with 0 hidden units (using the BERT embeddings directly into the classifier layer). This experiment is probably the closest to the ”fine-tuning” used in the paper, and the accuracy found was 0.865.

4.2.5 Number of Training Epochs

As expected, the more training epochs we used, the better the model performed. Figure 3 shows that we found that the performance stopped improving after about 50 epochs.

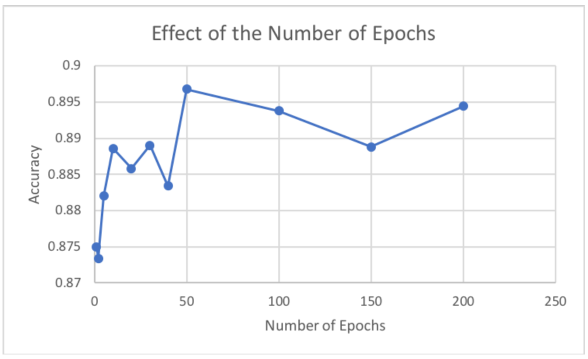

Figure 3: Effect of the Number of Epochs

4.2.6 Activation Function

Activation Function	Accuracy
ReLU	0.8930
Softmax	0.8726
tanh	0.8874

Table 5: Effect of Activation Function.

Table 5 shows that the ReLU activation function performs the best. ReLU has been found to converge faster and avoids the vanishing gradient issue, so this result is not too surprising (Glorot et al., 2011).

4.2.7 Dropout Rate

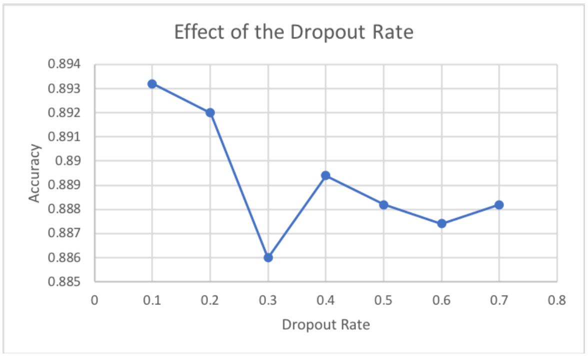

Figure 4: Effect of the Dropout Rate

Figure 4 shows that as we increase the dropout rate, the accuracy decreases, though not significantly. This may be a little biased since as the dropout rate increases, we will need more epochs for convergence. However, even after increasing the number of epochs, we found that the accuracy was about the same across all dropout rates.

4.2.8 Optimizer

Optimizer	Accuracy
AdaGrad	0.8930
Adam	0.8732

Table 6: Effect of optimizer used.

Table 6 shows that AdaGrad performs better.

4.2.9 Number of Training Examples

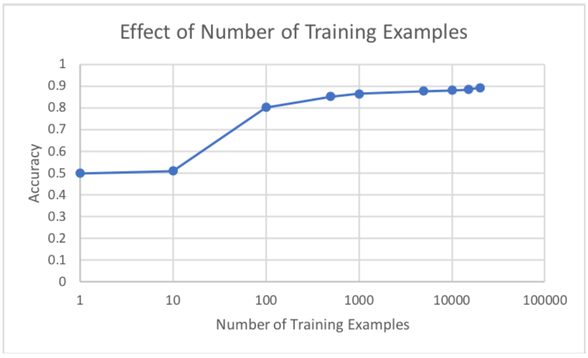

Figure 5: Effect of the Number of Training Examples

We also assess the effect of the number of training examples the model receives. As Figure 5 shows, the model performs better when as it receives more training examples, though the marginal benefit does seem to decrease with time.

4.2.10 Pooling Strategy

Pooling Strategy	Accuracy
None	0.7092
Reduce Max	0.6972
Reduce Mean	0.7126
Reduce Mean Max	0.7116
First Token	0.7168
Last Token	0.6988

Table 7: Effect of pooling strategy at the second to last layer. Note that these results are run with different parameters than the default ones and on the small model (hence the lower numbers)

First and last token embeddings in the BERT model are a special classification related to aggregate sequence representation (e.g. sentence pairs packed into a single sequence). Pooling with 'First Token' ("CLS" or "Classification" token; representations of sentences) performed better than 'Last Token' ("SEP" or "Separator" token; representations of multi-sentence sequences).

([CLS]) and ([SEP]) tokens are meant to be optimized with fine-tuning tasks which were not included in our training and therefore were not considered for final model performance. Overall, we found the default reduce mean strategy to work reasonably well.

4.2.11 Learning Rate

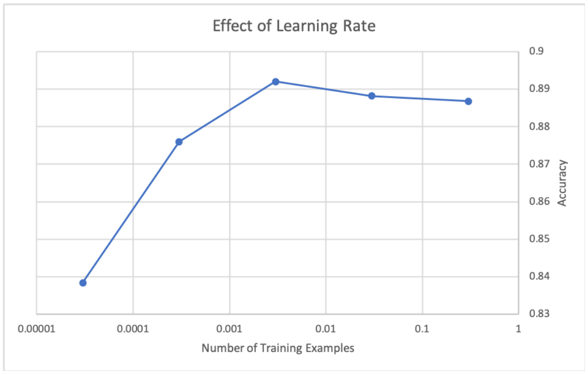

Figure 6: Effect of the Learning Rate

Figure 6 shows that accuracy stabilizes above an approximate learning rate of 0.003 and begins to fall as learning rate becomes smaller. As learning rates effect convergence, we use 0.003 for all further experiments.

4.3 Model Generalizability

To get a sense of how well our model generalizes the training data to the test data, we compare the accuracy of the model on the training data to the model on the testing data (using the default parameters). Table 8 shows that the model performs very well on the training data, and loses about 8% of performance when generalizing to the testing data.

Dataset	Accuracy
Training (20k)	0.9727
Development (5k)	0.8930

Table 8: Performance of model on training v. dev data.

4.4 Final Model

As a result of our human-guided explorations, we feel that the model with the default parameters set above is a reasonably good model for the IMDB sentiment analysis task. The only change we make is setting the training epochs to 50 rather than 30 since that was shown to work well. The remainder of the analysis performed in this section will be on that model.

4.5 Error Analysis

Looking at the mistakes that our model makes, we noticed that most of the errors happen on reviews that don't actually begin by mentioning the author's sentiment on the movie. For example, one review starts: "Before I begin..." and then goes on to describe his opinion of the director. Future experiments may consider weighting different sections of reviews differently.

4.6 Benchmarks

Method	Accuracy
Baseline (always guess positive)	0.5000
Google's fine-tuned BERT (small uncased)[4]	0.8665
(Ours) BERT embedding + Logistic Regression	0.8816
(Ours) BERT embedding + Linear SVM	0.8864
(Maas et al., 2011) Semantic + Semantic Embedding + linear SVM	0.8889
(Ours) BERT embedding + neural net classifier	0.8921
(McCann et al., 2017) Contextualized Word Vectors	0.9180
(Johnson and Zhang, 2016) oh-2LSTMp	0.9406
(Miyato et al., 2016) Adversarial Training	0.9409
(Gray et al., 2017) Sparse LSTM	0.9499
(Howard and Ruder, 2018) ULMFiT	0.9540

Table 9: Comparison with other models.

Table 9 shows the performance of our model against other paper's approaches toward this problem. Since this is a binary classification task with a 50/50 split in positive and negative examples, the baseline performance is 50%. In addition to other paper's works, we also show the results of feeding the BERT embeddings as input to a linear SVM and logistic regression. We do this to assess the value-add of putting a DNN layer on top of the BERT embeddings. For proper comparison in this section, we trained our network on the entire 25k training set and tested it on the 25k testing set.

While there are a couple of models that perform better than ours, our model performs respectably, beating Google's example BERT implementation as well as the original paper's implementation. Additionally, training and testing on the full datasets gives a similar result to that which we found when using a smaller training and development set (0.8930 v. 0.8921). Given more compute, we believe we could make major improvements to our existing model. For example, we could embed more of each review as well as use Google's fine-tuning architecture. In addition, we do see that the feeding the BERT embeddings directly into logistic regression and linear SVM both perform fairly well.

5 Conclusion

We trained a BERT model on a well-constructed, large movie review dataset and demonstrated an accuracy of 89.21%, well over the baseline of 50%. Through our tests with the effect of various parameters, we found that the BERT model size and number of tokens embedded were two of the largest factors influencing our performance, and that other factors, such as the number of training epochs and activation function matter, though to a lesser extent. While our model did not beat state-of-the-art performance, it performed comparably to the state-of-the-art models and better than (Maas et al., 2011)'s initial attempt and Google's example network. In the future, we would like to further explore embedding more tokens in a review, looking at embedding different parts of a review, and extending Google's architecture with our findings. Given our simple take with BERT embeddings has performed so well, we definitely believe that BERT embeddings has strong promise to produce state-of-the-art results in many NLP tasks.

Acknowledgments

We would like to thank Daniel Cer and the rest of the W266 staff for a great course!

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. 2011. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323.

Scott Gray, Alec Radford, and Diederik P Kingma. 2017. Gpu kernels for block-sparse weights. *arXiv preprint arXiv:1711.09224*.

Jeremy Howard and Sebastian Ruder. 2018. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*.

Rie Johnson and Tong Zhang. 2016. Supervised and semi-supervised text categorization using lstm for region embeddings. *arXiv preprint arXiv:1602.02373*.

Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. 2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1*, pages 142–150. Association for Computational Linguistics.

Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. 2017. Learned in translation: Contextualized word vectors. In *Advances in Neural Information Processing Systems*, pages 6294–6305.

Takeru Miyato, Andrew M Dai, and Ian Goodfellow. 2016. Adversarial training methods for semi-supervised text classification. *arXiv preprint arXiv:1605.07725*.

Yla R Tausczik and James W Pennebaker. 2010. The psychological meaning of words: Liwc and computerized text analysis methods. *Journal of language and social psychology*, 29(1):24–54.