Individual Final Report

Student: Srijon Mukhopadhyay
Instructor: Dr. Amir Jafari
Date:04/03/21

Introduction

Sentiment Analysis has been a very favored task especially in business sectors. It provides insights into customer trends and may be used for adaptive customer service and training chatbots. Using Sentiment Analysis, brands are able to assess customer reactions to their products and make important business decisions accordingly.

Deep learning models are very useful for making use of sentiment analysis tools. Both supervised and unsupervised approaches have been used in the past to implement sentiment analysis.

In this project we compared two popular models based on their ability to learn from labeled data and perform classification tasks - the CNN model and the LSTM model.

Description of Individual Work

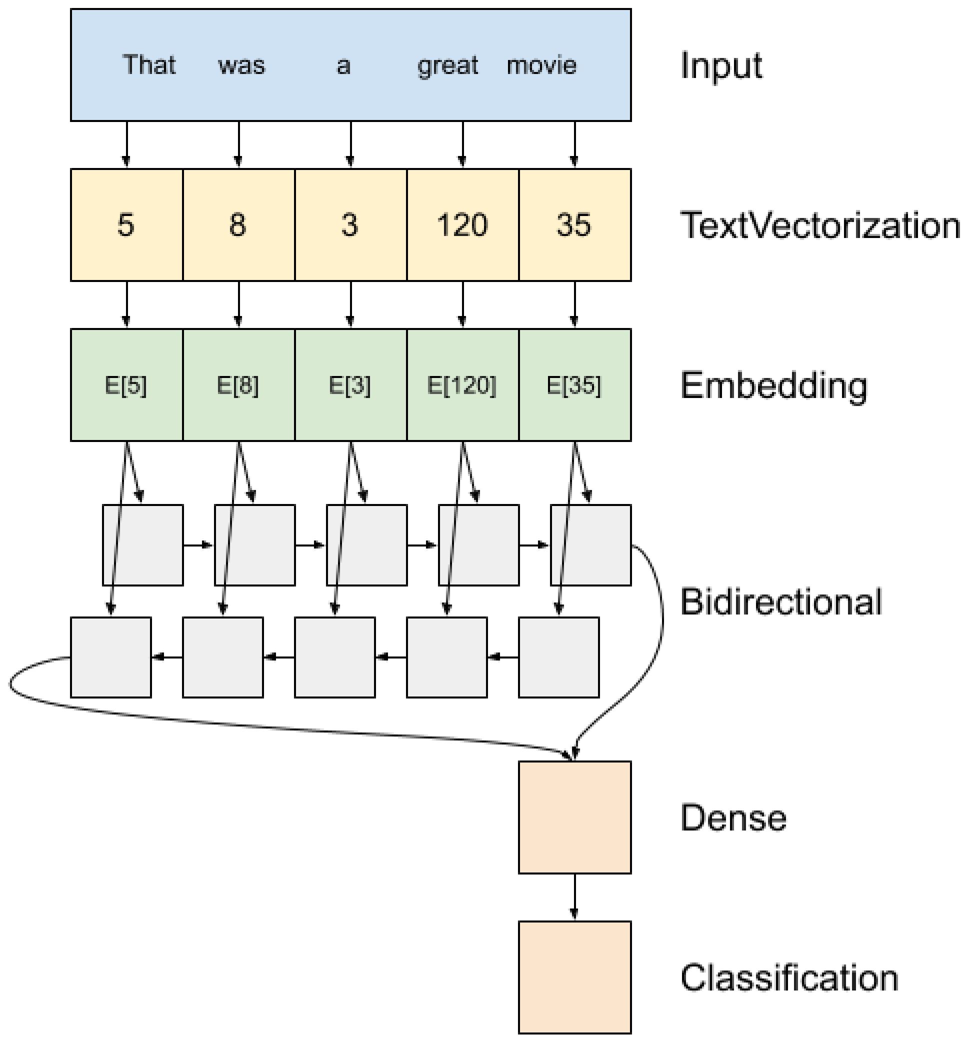
For my part in this project I created the LSTM algorithm used for predicting positive and negative sentiments and also evaluated the best fit of the CNN model and my LSTM model.

My partner had already downloaded the dataset and preprocessed it and made a simple perceptron model. She had also made a CNN model that was predicting positive and negative sentiments.

I downloaded the dataset on my google collaborator and ran her preprocessed code blocks. I then started developing the LSTM algorithm we used.

LSTM

A Long Short Term Memory (LSTM) model is a model that is typically used for text classification. It is a special type of an RNN model and is also able to save information about past inputs when processing the current input. Its advantage over the RNN model is that it is also able to retain information about long term dependencies. This solves the vanishing gradient problem that RNNs usually face. Since text data usually incorporates long term dependencies, this makes LSTMs very useful for solving problems such as text classification, sentiment analysis, information retrieval, etc.



sourced from: https://www.tensorflow.org/tutorials/text/text_classification_rnn

Since this is a binary classification task, the models had a final sigmoid activation layer which will output a value between 0 and 1. A value greater than 0.5 will be considered to be a positive sentiment while a value less than 0.5 is considered to be a negative sentiment.

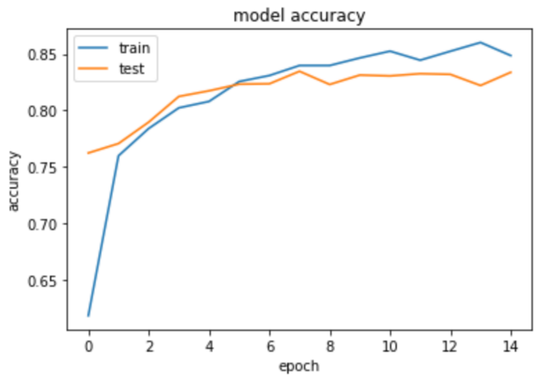
The parameters that I updated mostly were minibatch size and learning rate. My primary readings online suggested that higher batch sizes generally tend to have lower accuracy when the model is evaluated on the test set. I also read that this lost test accuracy may be recovered by increasing the learning rate.

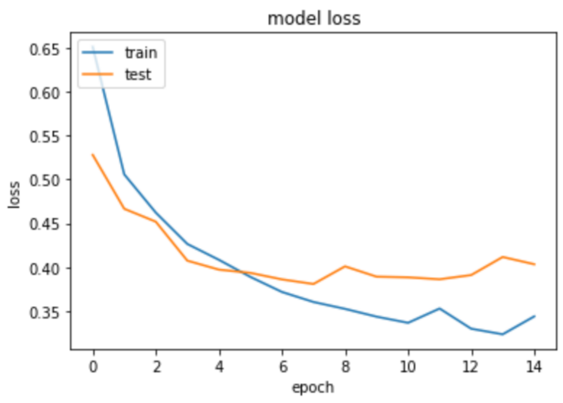
We thus tried for two different cases in our models:

- Fits with higher batch sizes and learning rates
- Fits with lower batch sizes and learning rates

Once the data is fit I evaluated the model by making predictions on the test set and evaluating the accuracy. I also used my own metrics to assess the performance of our models.

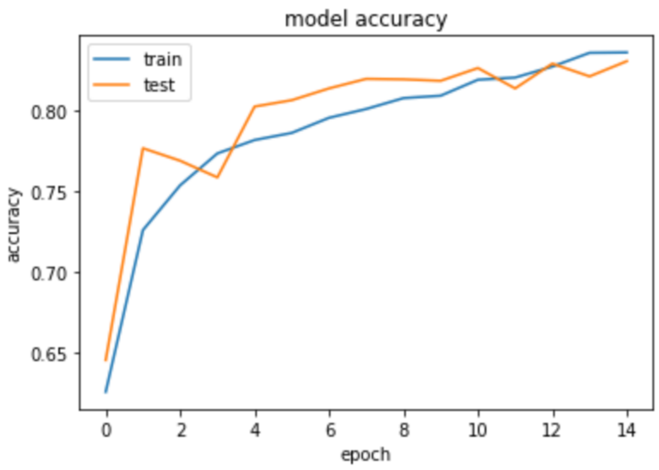
The main goal was to find an optimal ratio between batch size and learning rate. I tried a higher batch size coupled with high learning rate:

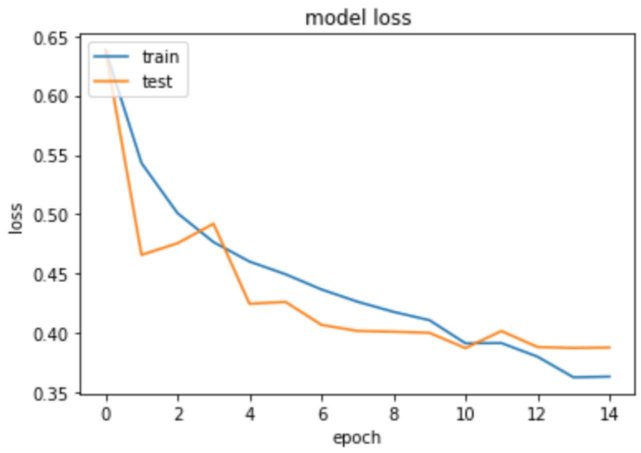




This gave us an accuracy of 0.83.

Next I tried another fit with lower batch size (32) and learning rate (0.001):





This had an accuracy of 0.84 and the fit seems to improve.

To prevent overfitting for the LSTM, a dropout layer of 0.3 was implemented along with recurrent dropouts of 0.2 so they wouldn't affect the final dense layer.

Though overfitting was still not controlled to the best possible rate in the LSTM model there was a constraint that we had to face. The LSTM model in general requires a lot of data points to train. Since we were limited with our training samples, adding higher dropout layers to the model would affect the fit of the model as well.

Results

I was responsible for evaluating both the CNN model and LSTM models. I evaluated the best fits of our CNN and LSTM model mainly using three metrics:

- F1 Scores, Precision and Recall

- Confusion Matrix
- AUC curve

For CNN:

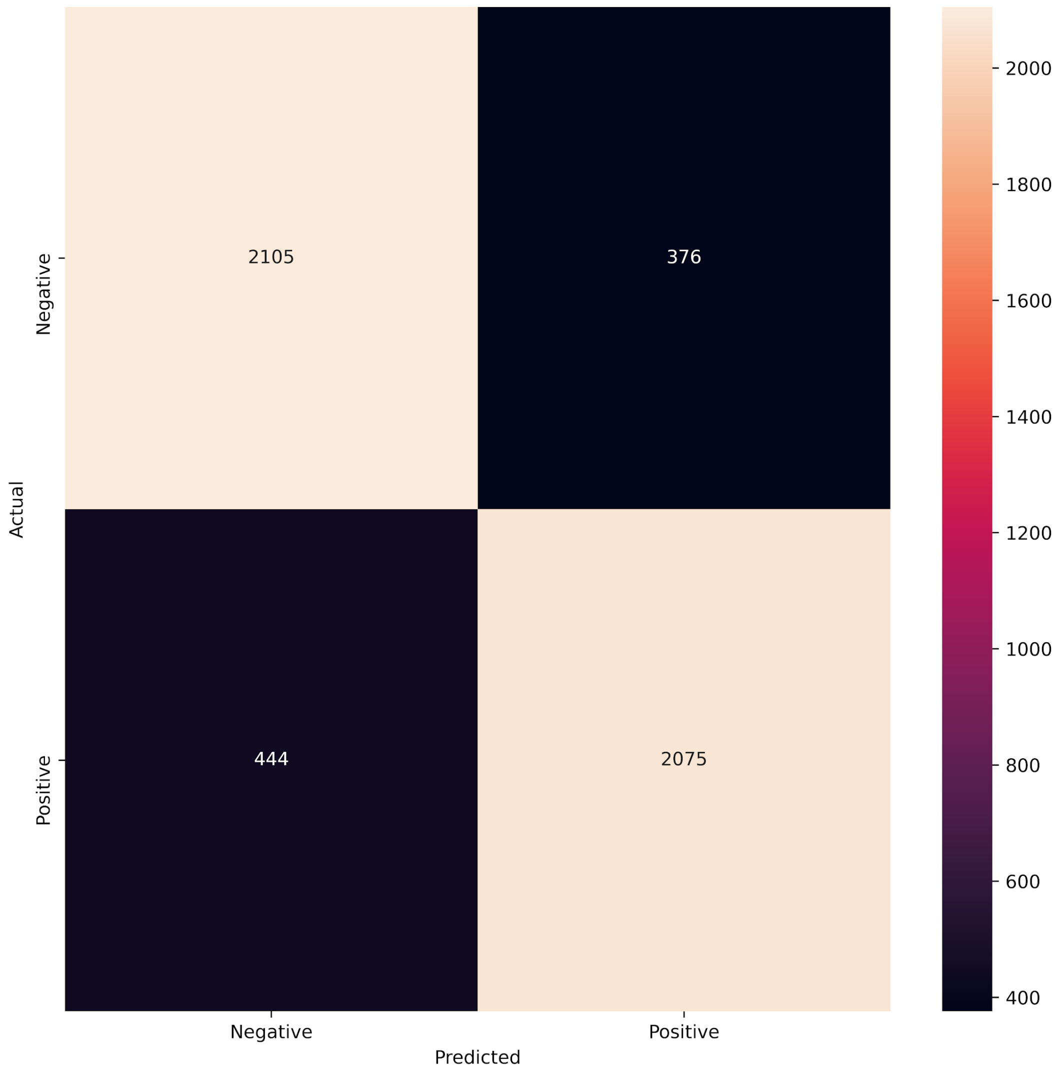
Sentiment(CNN)	Precision	Recall	F1 Score
Negative	0.83	0.85	0.84
Positive	0.85	0.82	0.84

Precision scores impies the proportion of the predicted instances that are actually relevant or predicted correctly, while Recall implies the proportion of relevant instances that the model is able to predict correctly. Since both precision cannot be improved without decreasing Recall, it is important to have another metric that provides a measure of how the model does in both cases. The F1 score is useful in this case and is the harmonic mean of the precision and recall scores of the model.

A high F1 score would imply that the model is able to control for a high a number of false negatives and false positives and can thus make reliable predictions.

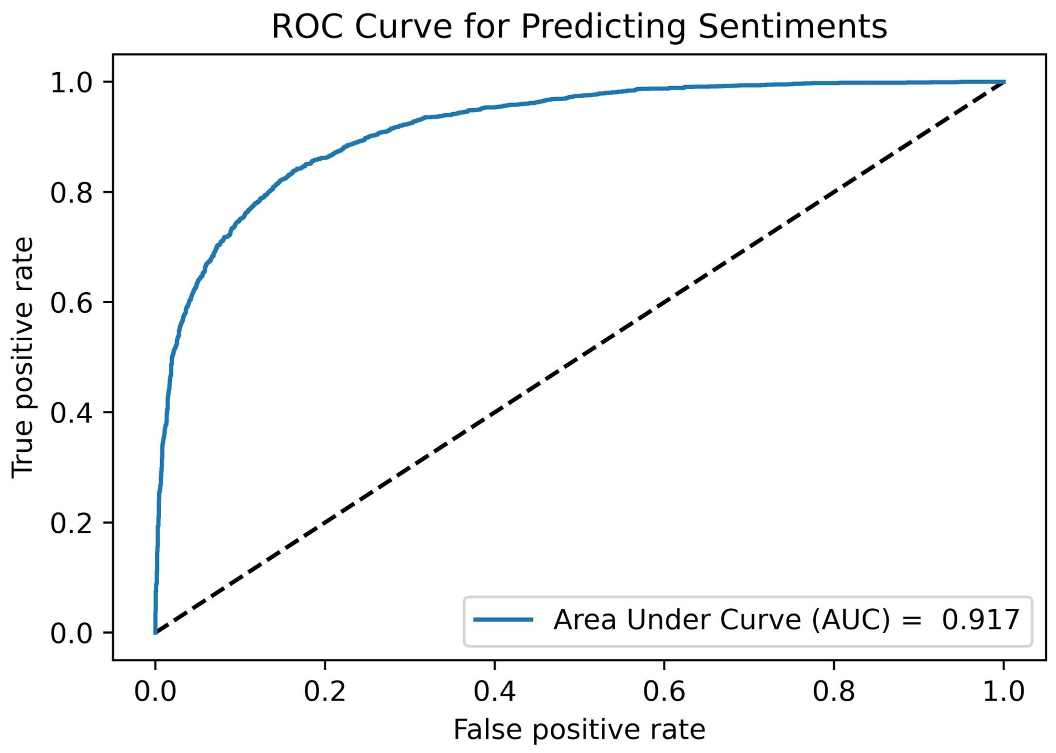
The CNN model had an F1 score of 0.84 for both the negative and positive sentiments which implies that there was no imbalance and one sentiment wasn't detected unfavorably over the other by the model.

For this type of problem, we would assume that there would be more emphasis on not mistakenly predicting negative sentiments as positive sentiments than the opposite. Especially in the case of business sectors, it is important to identify negative sentiments correctly so as to make better business decisions that will keep customers happy. We thus want to have fewer false positives for predicting positive sentiments than false negatives.



Looking at the confusion matrix, negative sentiments were predicted correctly 2105 times out of a total 2481 instances. Positive sentiments were predicted correctly 2075 times out of 2519 total instances.

The CNN model thus has 376 false positives when it attempts to predict positive sentiments.



Next, we plotted an AUC-ROC curve. The ROC curve, a performance measurement, is a probability curve between the True Positive Rate and False Positive Rate. We made the AUC-ROC curve based on how many times our model predicted positive sentiments.

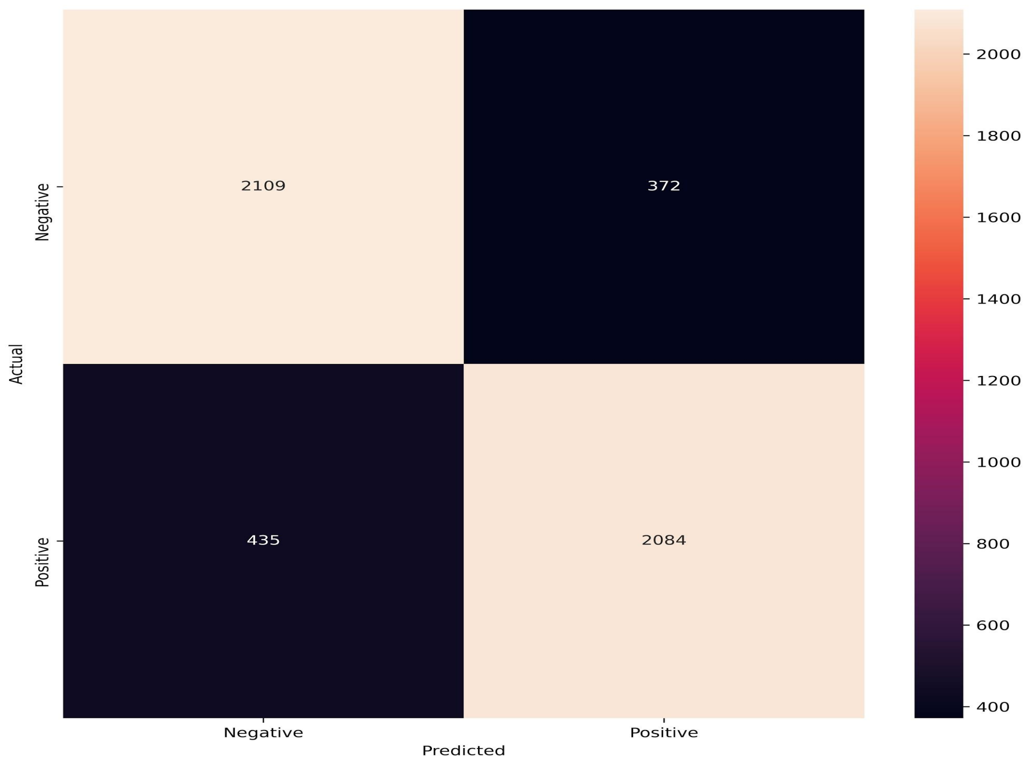
The AUC is the area under the curve of the ROC curve. Essentially, it is a degree of separability and its values can range between 0 and 1. A value of 0.5 would mean that the model is not differentiating at all between the two classes and 1 would mean that the model is perfectly distinguishing between positive and negative cases. When the value is 0, it means the model is predicting negative sentiments as positive sentiments and vice versa.

Our CNN model has a AUC of 0.917 which implies a 91% degree of separability between positive and negative sentiments. This is a good score.

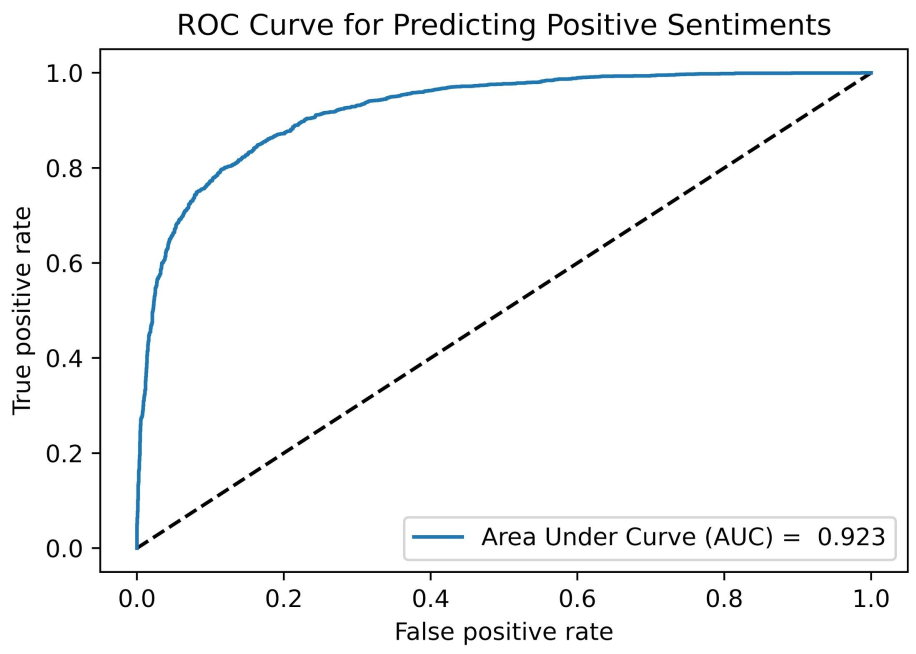
For LSTM:

Sentiment (LSTM)	Precision	Recall	F-1 Score
Negative	0.83	0.85	0.84
Positive	0.85	0.83	0.84

The LSTM model also displayed the same F1 scores as the CNN model that my partner made.



Though the F1 scores for both the CNN and LSTM models are the same, the LSTM model has 372 false positive errors when it attempts to predict positive sentiments from the test set. It thus performs better than CNN in the proposed context of the problem.



The AUC of the LSTM model is 92%. This is slightly more than the AUC of the CNN model, implying a better performance in being able to distinguish between positive and negative sentiments from the test dataset.

Conclusions

The LSTM model slightly outperforms the CNN model with it having:

- Lesser number of false positive errors
- Higher AUC scores
- Lesser number of false negative errors (though this was not of high priority)

Both LSTM and CNN models performed better with lower batch sizes and learning rates. Even though we followed our previous readings and fixed for the higher batch sizes by increasing their corresponding learning rates, the models still fit the training data better when the batch sizes and learning rates were lower.

This is probably because the higher learning rates do not offset the problem of higher batch sizes well. Higher batch sizes imply random gradient updates which may be very small or very large, all depending on the sample. Furthermore, if the total loss is averaged over the batch, then larger batch sizes do not offer the model much flexibility in traveling from its initial weight updates.

In contrast, the smaller batch size almost acts as a regularizing feature, allowing for uniform gradient updates and allowing more flexibility for models in updating weights. The lower learning rate is also better for converging to an optimum and prevents the risk of overshooting the point, as higher learning rates sometimes do. This also helps prevent overfitting and thus leads to better accuracies on the test set.

This project can be improved. Once major issue that we ran into was our model overfitting on the training data. This was not something we were able to control for as there weren't enough training data samples for us to increase our dropout without affecting model performance. One way to improve our model performance would be to increase the number of training samples.

More experimentation with hyper parameter tuning is also required. To improve this, it may be worthwhile to consider different ways to change the learning rates while the epochs are running (decays such as time decay, exponential decay, etc). For this task we used Adam as the optimizer, which is an adaptive optimizer. Since Adam updates the learning rate by itself, we ultimately did not go ahead with any decaying but it would also be interesting to use another optimizer like Stochastic Gradient and try different methods to lower learning rates while calibrating momentum values.

Around 41% of my code was copied from the internet.

References

https://wiki.pathmind.com/accuracy-precision-recall-f1

https://towardsdatascience.com/multi-class-text-classification-with-lstm-1590bee1bd17

https://machinelearningmastery.com/predict-sentiment-movie-reviews-using-deep-learning/

https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e

https://arxiv.org/pdf/1711.00489.pdf