Capstone Project

Sentiment Analysis using LSTM

Machine Learning Nanodegree

Anirudh Ramanan
31st July 2018

I. Definition

Project Overview

Sentiment Analysis is a process of determining whether a given statement is positive, negative or neutral. A basic task in **sentiment analysis** is classifying the polarity of a given text of the document, sentence, or feature/aspect level—whether the expressed opinion in a document, a sentence or an entity feature/aspect is positive, negative, or neutral. A common use case for this is to determine what people think or feel about a particular topic, or helping a business to understand the social sentiment of their brand, product or the services they offer by monitoring the online conversations.

As the Natural Language Processing techniques keeps on improving and computational power gets cheaper, and with availability of internet with good training dataset, it has become much easier and more efforts are being put into automated text processing methods.

Sentiment Analysis makes use of text mining and NLP in order to identify and extract the context by analysing the opinion, attitudes and emotion. Sentiment Analysis can be applied to various fields / problem statements such as analysing a movie review, or for stock prediction. Stock prediction is one of the major field where sentiment of the company plays a major role in driving the prices.

The aim of the project is to analyse the sentiment of a given movie review.

The dataset has been downloaded from http://ai.stanford.edu website. This dataset contains movie reviews along with their associated binary sentiment labels. The

core dataset contains 50,000 reviews (25K positive and 25K negative, balanced distribution) split evenly into 25K training and 25K test sets.

Project Statement

The aim of the project is to correctly label the movie reviews as positive or negative i.e based on the word features of the given review, correctly classify them as a positive review or a negative review.

The solution to solve this problem is to create word embeddings out of the review words, and then feed the neural network for training.

What is word embedding ?

As it is not possible to perform back-propagation and other tasks on a series if strings, the reviews/sentences needs to be converted into a vector format which can then be fed into the neural network for training. Its input is a text corpus and its outputs a set of vectors i.e it turns text into numerical form that the neural network can understand.

This is what the overall solution will look like:

- Data Preprocessing
    - Cleaning data
    - Converting words into vectors
- Splitting training set to train and validation set
- Build the LSTM Graph
- Training
- Testing

Metrics

Accuracy Score

Accuracy Score of the model will be one of the metrics against which the performance of the model will be evaluated. Since the aim is to correctly classify

the sentiment of a given statement, accuracy score is one of the right metric with which the performance of the model can be evaluated.

False Positives

Since the test set is already categorised into positive and negative sections, we can cross verify the result which is predicted by the model against the labelled test set. Using this we can easily determine the number of false positives in the predicted result.

II. Analysis

Data Exploration

The dataset has been downloaded from http://ai.stanford.edu website. This dataset contains movie reviews along with their associated binary sentiment labels. The core dataset contains 50,000 reviews (25K positive and 25K negative, balanced distribution) split evenly into 25K training and 25K test sets.

This means the training dataset and test dataset contains 12.5k positive and 12.5k negative reviews.

In the entire collection, no more than 30 reviews are allowed for any given movie because reviews for the same movie tend to have correlated ratings. Further, the train and test sets contain a disjoint set of movies, so no significant performance is obtained by memorising movie-unique terms and their associated with observed labels. In the labeled train/test sets, a negative review has a score <= 4 out of 10, and a positive review has a score >= 7 out of 10. Thus reviews with more neutral ratings are not included in the train/test sets. In the unsupervised set, reviews of any rating are included and there are an even number of reviews > 5 and <= 5.

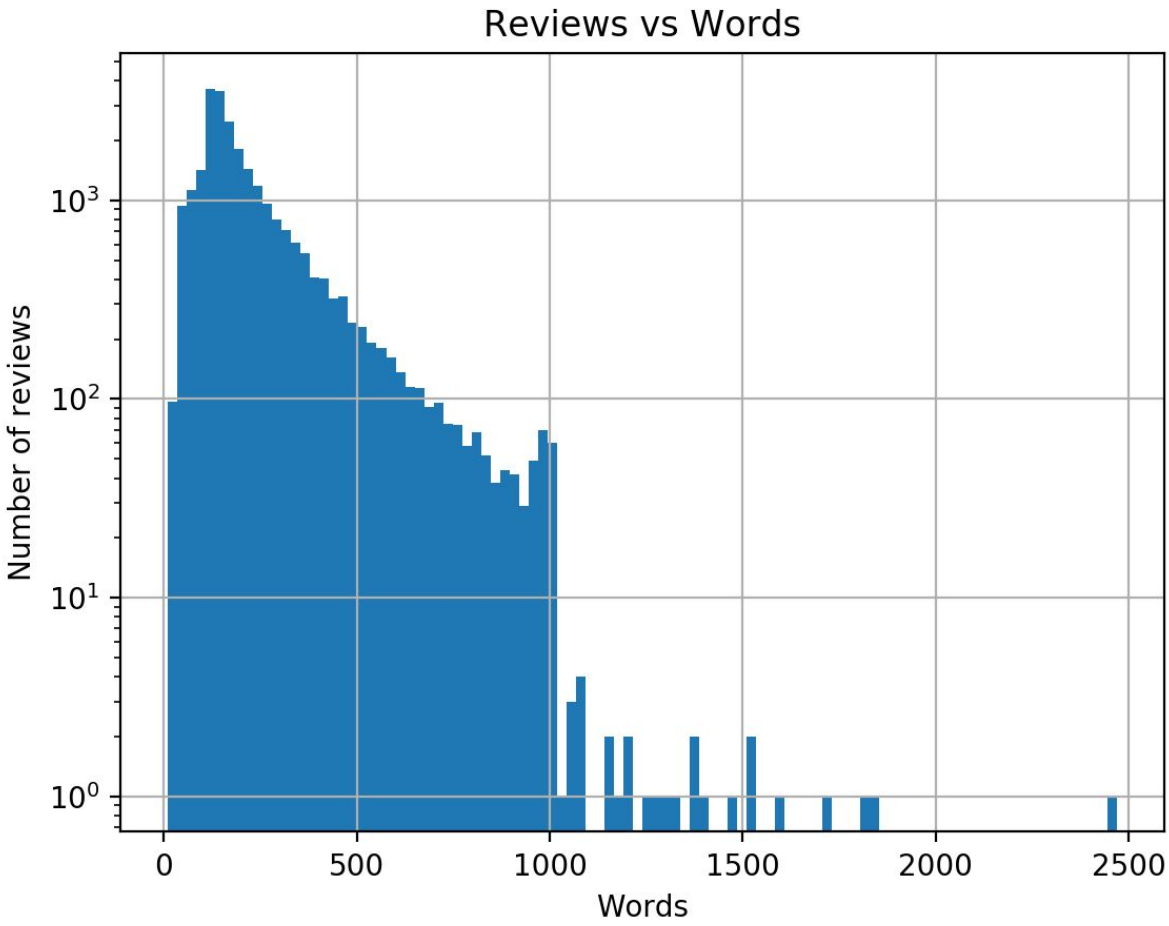
Using the files in the dataset, I have created a train.csv file which contains two features (one is the file_path, and second is the target ie 0 for negative and 1 for positive), which will be used to load the dataset. We will loop through the csv file,

and load the corresponding review file which is then added to the review array for creating the word embedding.

The dataset does not contain any abnormalities such as missing values, or outliers.

Exploratory Visualisation

Let's start by plotting the number of words in all the reviews:

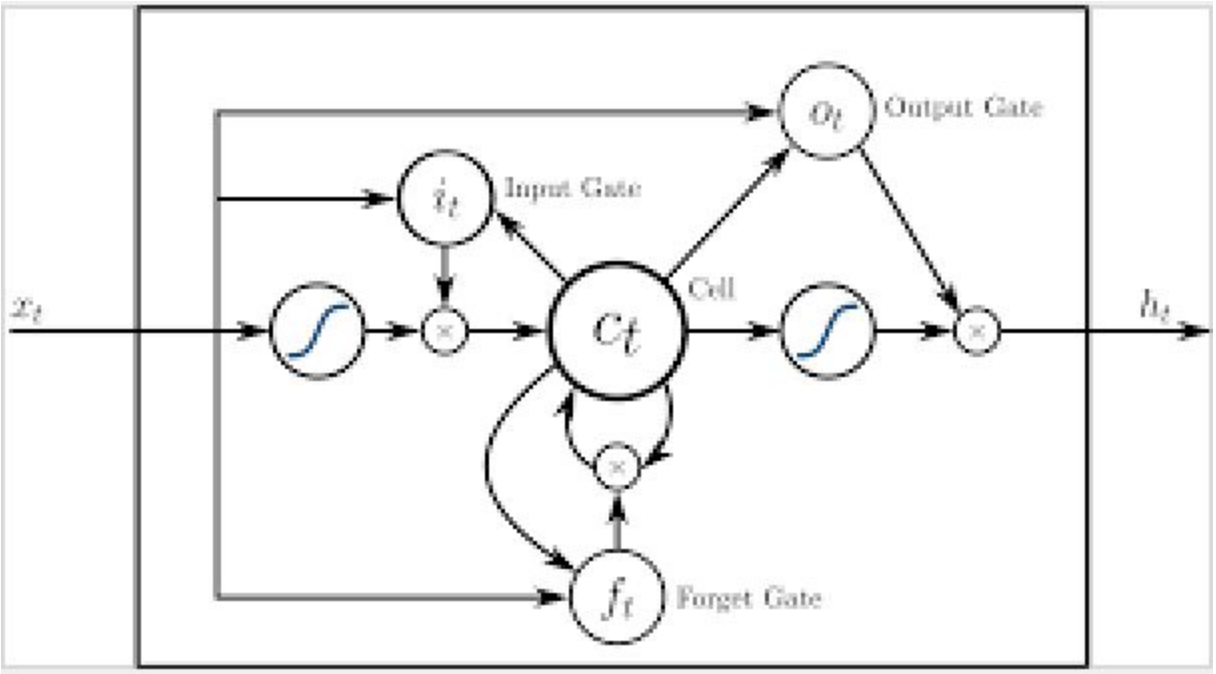


The graph shows the words count in all movie reviews. As we see, most of the reviews contain approximately 100-1000 words. As an outlier, one of the review has ~2400 word count.

Algorithms and Techniques

Long Short Term Memory Neural Network (LSTM)

LSTMs are a special kind of RNN which are capable of learning and storing long term dependencies.



A regular LSTM consists of a cell, an input gate, an output gate and a forget gate. The cell is primarily responsible for remembering values over arbitrary time periods. The gates compute an activation using the logistic function. The input gate controls how much information flows into the cell, the forget gate controls the extent to which a value remains in the cell and the output gate controls the extent to which the value in the cell is used to compute the output activation of the LSTM unit.

We will be using tensorflow graph api's for building the LSTM graph end to end. Following are the hyper-parameters that will be used to build the graph:

- LSTM-size : Number of hidden layers in the LSTM cell. The larger the size, the better the performance. Most common values that are used are : 128, 256, 512.

- LSTM-layers: Total number of LSTM layers in the neural network

- Learning-rate: Self-Explanatory

- Batch-size: Number of reviews to be fed into the neural network in each training step. This depends on the environment being used for training, and should be tweaked as per the resources available.

- Embed-size: size of the embedding layer

Embedding Layer:

As it is not possible to perform back-propagation and other tasks on a series if strings, the reviews/sentences needs to be converted into a vector format which can then be fed into the neural network for training. Its input is a text corpus and its outputs a set of vectors i.e it turns text into numerical form that the neural network can understand. An embedding is a mapping from discrete objects, such as words, to vectors of real numbers. To create word embeddings in TensorFlow, we first split the text into words and then assign an integer to every word in the vocabulary.

LSTM Cell:

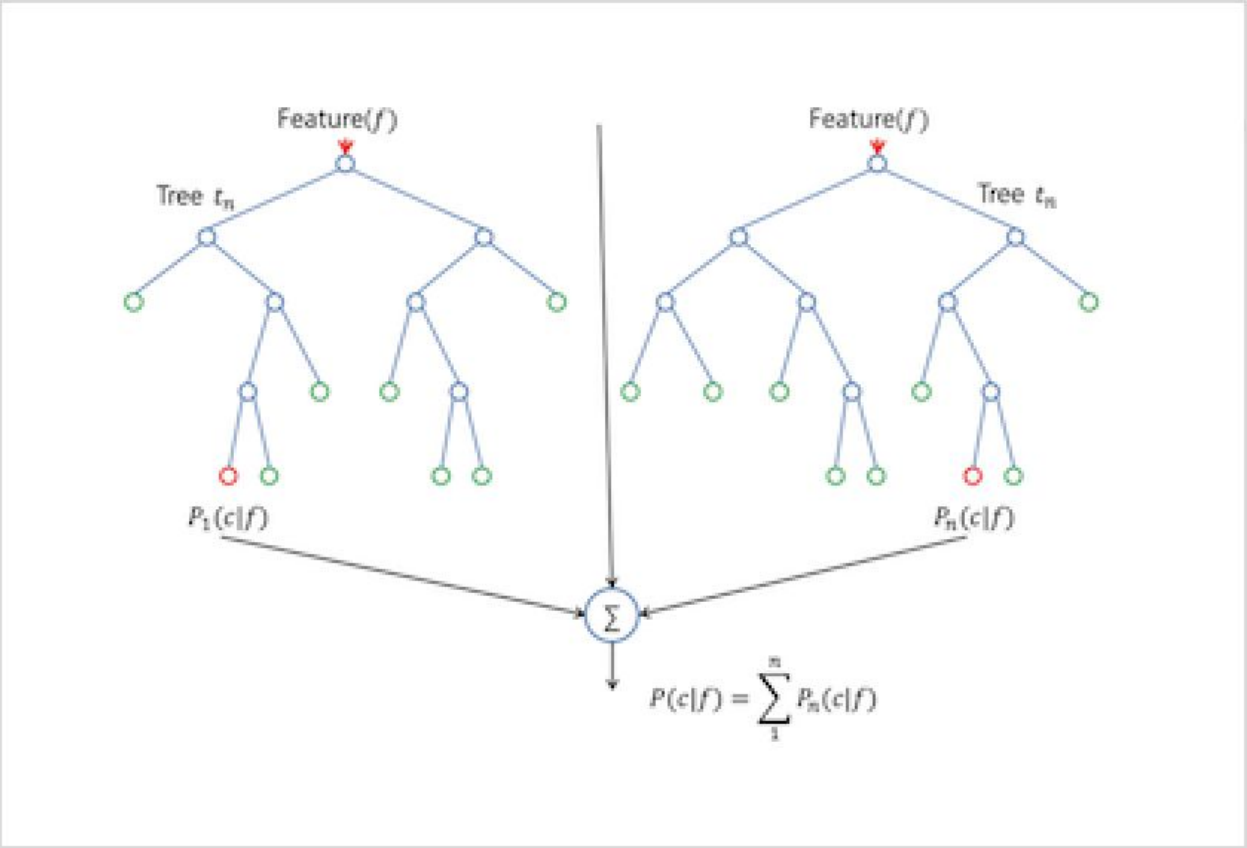
Next, we define the Long short-term memory unit (LSTM) recurrent network cell which is used in the network. The more number of layers, the better the performant. Saying that increasing the number of layers beyond a certain point may overfit the dataset and will give lower accuracy.

Dynamic RNN:

We need to actually run the data through the RNN nodes. We pass in our cell and the input to the cell, then it does the unrolling and everything else for us. It returns outputs for each time step and the final_state of the hidden layer.

Benchmark

Random Forest Classifier is used as a benchmark model which will be trained on the vectors created from the words in the sentences, which in turn will be used to predict and classify the sentiment of a review.



III. Methodology

Data Preprocessing

The dataset contains movie reviews along with their associated binary sentiment labels. The core dataset contains 50,000 reviews (25K positive and 25K negative, balanced distribution) split evenly into 25K training and 25K test sets. Using the files in the dataset, I have created a train.csv file which contains two features (one

is the file_path, and second is the target ie 0 for negative and 1 for positive), which will be used to load the dataset.

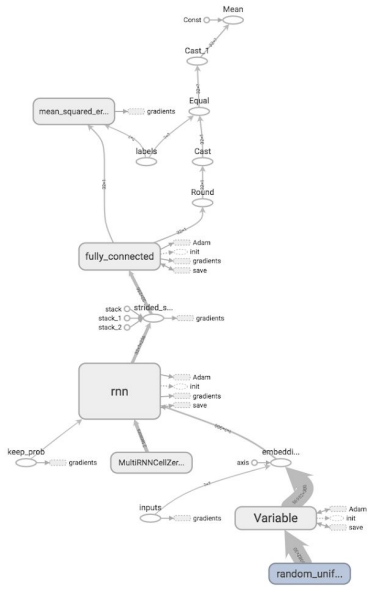
The dataset does not contain any abnormalities such as missing values, or outliers.

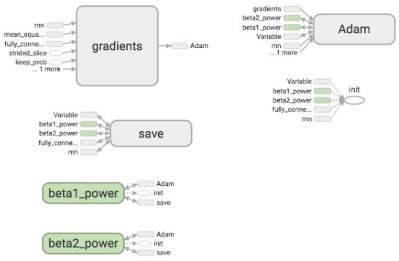
Implementation

After the data preprocessing stage, we need to convert the words into the vector representation. Why is it required ? As explained earlier, neural networks cannot perform back-propagation and other tasks on the string input, hence before feeding it into the neural network, the words of the sentences have to be converted into the vector format, also known as Word Embedding.

Neural Network

The embedding lookup requires that we pass in integers to the network. The easiest way to do this is to create dictionaries that maps the words in the vocabulary to integers. For this we will use Counter class of the python library. A **Counter** is a subclass for counting hashable objects. It is an unordered collection where elements are stored as dictionary keys and their counts are stored as dictionary values.





Once we have the words encoding, we can feed this into the neural network. The neural network built using the tensorflow api's contains a single LSTM cell with 512 hidden layers. The training is done for 10 epochs with a learning rate of 0.001. The reviews/word embedding vectors are fed into the neural network in batches of 500, which means for one single epoch the total number of iterations will be (25000/500)=50 iterations. The accuracy is calculated by taking out the mean value of the accuracy of the each iteration. Once the training is completed, the session is saved to a checkpoint.

Random Forest Classifier

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. Random Forest Classifier is used as a benchmark model which will be trained on the vectors created from the words in the sentences, which in turn will be used to predict and classify the sentiment of a review.

Refinement

The initial training on the given dataset gave a very low accuracy rate. After some hit and trial, and learning more about how to pre-process the data, I was able to tweak the model for a good accuracy rate.

- Initially the dataset contained 12,500 k rows of positive sentiment first, and then 12,500 k rows of negative sentiment. Due to this the neural network used to give accuracy of 100% since the label/target used for learning remained the same for first 12,500 rows. Now, the dataset has been shuffled to include a mixture of both positive and negative reviews.
- The batch size: It is true that keeping a very low batch size might also have an impact on the accuracy rate. So I had to tweak the batch size keeping in mind that the model does not run out of resources while training, as well as the accuracy is at par.
- Number of LSTM cells: Initially with 2 LSTM cells with 512 hidden layers, the accuracy did not improve as expected. The reason could be overfilling of the data. Hence the network now uses a single LSTM cell with 512 hidden layers.

IV. Results

Model Evaluation and Validation

The model has a training accuracy for **97.8857**%. The training set contained equal number of positive and negative reviews. The final parameters which were used to train the model were:

- Lstm-layer: 512
- Lstm-cell: 1
- Learning Rate: 0.001
- Batch-size: 500
- Embed-size: 300

```
Batch: 36  | Loss: 0.008798755705356598  | Accuracy: 0.9900000095367432
Batch: 37  | Loss: 0.02301359921693802  | Accuracy: 0.9779999852180481
Batch: 38  | Loss: 0.0235122200101614  | Accuracy: 0.9760000109672546
Batch: 39  | Loss: 0.01755458302795887  | Accuracy: 0.9819999933242798
Batch: 40  | Loss: 0.014984719455242157  | Accuracy: 0.984000027179718
Batch: 41  | Loss: 0.023399757221341133  | Accuracy: 0.972000002861023
Batch: 42  | Loss: 0.015470970422029495  | Accuracy: 0.9819999933242798
Epoch: 24999/10  | Current loss: 0.019214371219277382  | Training accuracy: 97.8857
```

Validation test ie (0.15% of the training set) contains 3125 reviews. The validation set has an accuracy of **0.840** using the above trained model.

```
2018-07-31 08:11:44.460690: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1471] Adding visible gpu devices: 0
2018-07-31 08:11:44.460770: I tensorflow/core/common_runtime/gpu/gpu_device.cc:952] Device interconnect StreamExecut
2018-07-31 08:11:44.460803: I tensorflow/core/common_runtime/gpu/gpu_device.cc:958]      0
2018-07-31 08:11:44.460830: I tensorflow/core/common_runtime/gpu/gpu_device.cc:971] 0:   N
2018-07-31 08:11:44.460997: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1084] Created TensorFlow device (/job
Test accuracy: 0.840
```

Since the model has been trained on limited number of word sets, it might run into problem when it encounters a new word, or a misspelled word. The only solution for this is to train the model on all possible data-sets which includes all kinds of combinations and permutations.

I could have used the pre-trained model (Word2Vec) for this project, but I decided to create my own word embedding layer out of curiosity.

Justification

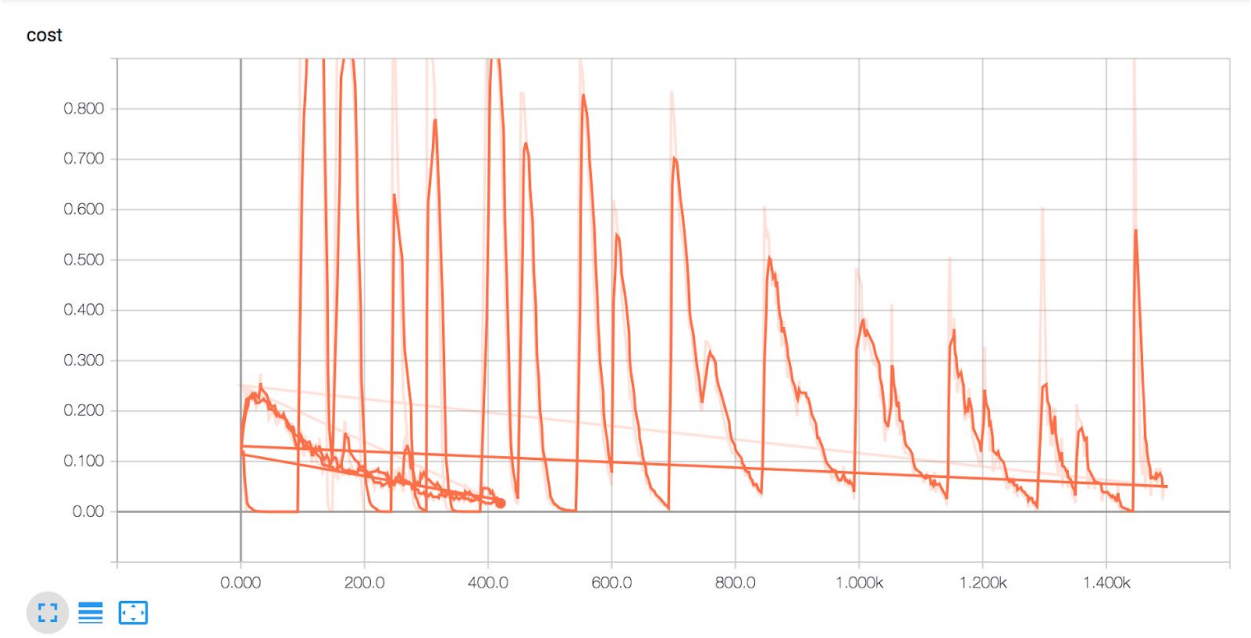
The benchmark model ie Random Forest Classifier received a test accuracy of **0.5261.** The reason could be that the classifier may not be able to store and process the context of the statement. The LSTM model performed ~38% better in terms of accuracy compared to the forest classifier model.

Hence, the vectors that were created from the movie reviews do not perform very well on the benchmark model as we thought.

V. Conclusion

Free-Form Visualization

Below is the graph which shows the cost occurred during the training process against the steps.



During the initial phase of training, the cost incurred by the model is a bit high, the reason which is self explanatory. As the training proceeds and learns the word vectors, the cost keeps on declining.

Reflection

The idea of the overall project was to examine whether LSTM network would perform better on the sentiment analysis dataset, and also examine whether models like Random Forest Classifier fits better on such problems. As I moved to the project, and with the first implementation of LSTM model, what i thought interesting is how well the LSTM network fits into the overall statement, and how it can store and learn long term dependencies.

The interesting aspect of the project was to create word embeddings out of the review dataset. This was also one of the challenging tasks that i encountered across. While reading about how LSTM and backpropagation works in the neural network, this all made sense.

The difficult part of the project was to choose the model which would fit best. This included a lot of research to come up and justify why this model would be the best fit. Also, since the word embeddings that I created has been trained on limited number of words, there is a possibility that this might not work very well when it encounters a new data-set, or a new word. Using Word2Vec could have solved this problem, but I decided to continue building my own word embedding layer.

Improvement

There are a lot of improvement which needs to be made to improve the efficiency of the model

- Using word2vec: I would later explore how Word2Vec would have performed in my case. Since creating word embedding on a limited data set may not fit well in future.

- Model Tweaking: I am thinking of writing a script which will train the model on different sets of hyperparameters everytime, and records the accuracy. This will help in finding the best hyperparameters which can give the best

result.

- Training the model on a large corpus of words.

VI. References

LSTM Wiki : https://en.wikipedia.org/wiki/Long_short-term_memory

Movie Dataset Credits : http://ai.stanford.edu/~amaas/data/sentiment/

O'reilly LSTM : https://www.oreilly.com/learning/perform-sentiment-analysis-with-lstms-using-tensorflow

RNN Wiki : https://en.wikipedia.org/wiki/Recurrent_neural_network

Udacity Deep Learning: https://github.com/udacity/deep-learning