CSE 676

Deep Learning

Sentiment Analysis Using RNN with LSTMs

-Arun Sharma(50206920)

-Aman Malkar(50206941)

Introduction:

- Sentiment analysis can be thought of as the exercise of taking a sentence, paragraph, document, or any piece of natural language, and determining whether that text's emotional tone is positive, negative or neutral.
- Our Project performs the task of Sentiment Analysis on a Movie Review [5] Dataset with the objective of training a model that would be able to classify reviews as Positive / Negative.
- Task (T) – Creating a Model (Classifier) that would determine if a review has a positive or negative polarity.
- Experience € – 25,000 Reviews in the form of text files containing 12,500 Positive Reviews and 12,500 Negative Reviews that will be used to train the Classifier.
- Performance (P) – Accuracy of the model to currently classify reviews as Positive or Negative.

Approach:

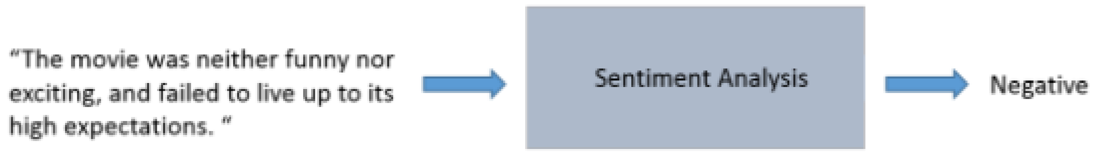
"The movie was neither funny nor exciting, and failed to live up to its high expectations. "
Sentiment Analysis
Negative

Fig 1: Sentiment Analysis Data Flow [2]

Data

- We want to build a model that would read a review, perform sentiment analysis and tell us whether it is positive or negative. But we cannot do operations like taking a dot product or back propagation on a string.
- That is, instead of processing strings, we need to have a vector representation of the words in the string on which we would perform sentiment analysis.
- Also, we need to have such a representation of the words that vectors of the words that have similar context, meaning and semantics be equivalent

to each other. To achieve this, we are using Word2Vec matrix trained using GloVe which is a word vector generation model. Our matrix has 400,000 word vectors each with a dimensionality of 50.

Recurrent Neural Networks

- We will be using Recurrent Neural Networks because RNNs associate each word of the input sequence with a specific time step and a hidden state vector. This hidden vector will encapsulate and summarize all the relevant information from the previous time steps.
- The activation function of the hidden state layer is a function of current word vector and hidden state vector of the previous time step. Thus, magnitude of the current word vector is affected by hidden vector of the previous states.
- The weights are updated through back propagation and the state vector at the final time step is fed into a binary softmax classifier where it is multiplied by another weight matrix and put through a softmax function that outputs values between 0 and 1, effectively giving us the probabilities of positive and negative sentiment [2].

Long Short Term Memory Units(LSTMs)

- Inside our RNN we will place a LSTM unit that will make it possible to determine what information should be retained in the hidden state vector.
- LSTMs have input, forget, output gates and a memory container. The input gate takes in the input of previous computation. Forget gate is used to throw away any data that it deems unessential for future use and output gate contains information to be fed in the next iteration.

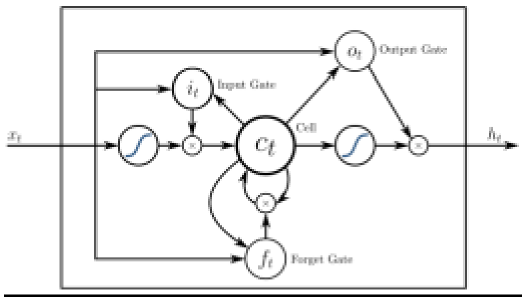

Fig 2: LSTM unit [4]

<u>Methodology</u>:

1. *Training and loading a word vector generation model (Word2Vec)*

We first load the dataset and create word2vec matrix and feed into training the model. The dataset has a fixed batch size, not too large in order to avoid computational complexity. Also has fixed number of LSTM units and iterations.

2. *Creating an ID's matrix for training set.*

Creating matrices with indices while comparing it to original Word2Vec model for the training set.

3. *Tuning RNN with LSTM model.*

Finally tune the model on the basis of number of iteration and batch size keeping in mind the trade-off between accuracy and time complexity. While tuning we used output of BasicLSTMcell function multiplying with weights and adding bais. The output is passed through softmax function which gives final predicted value.

$$Prediction = softmax(Wx + b)$$

Before passing the dataset into the model we first use dropout method in order to avoid overfitting and then use tf.nn.dynamic.rnn function in order to unroll whole network and create a pathway for data to flow through the RNN graph. For optimization, we used Adam Optimizer.

4. *Training the model*

For training we select random vectors from the 25,000 reviews of fixed batch size and saved the model on the disk for testing purposes and change the iteration for increasing accuracy in the test cases with reasonable time complexity. The output graphs are fetched from tensorboard.

5. *Testing the trained model*

Finally, we tested the trained model by selecting random vectors from the reviews set (as done in training) and find accuracy by taking the average accuracy values for 10 batches.

RESULTS

We passed the dataset while tuning the model through various hyperparameters such as batch size, iterations and LSTMSs units. Also, we set the learning rate as 0.001 (default) and used Adam Optimizer as directed in the paper.

Batch Size = 24
LSTM units = 64
Below are the training and testing graphs on the bases of the above parameters with varying their iterations.

Training Graphs

Iteration = 1000

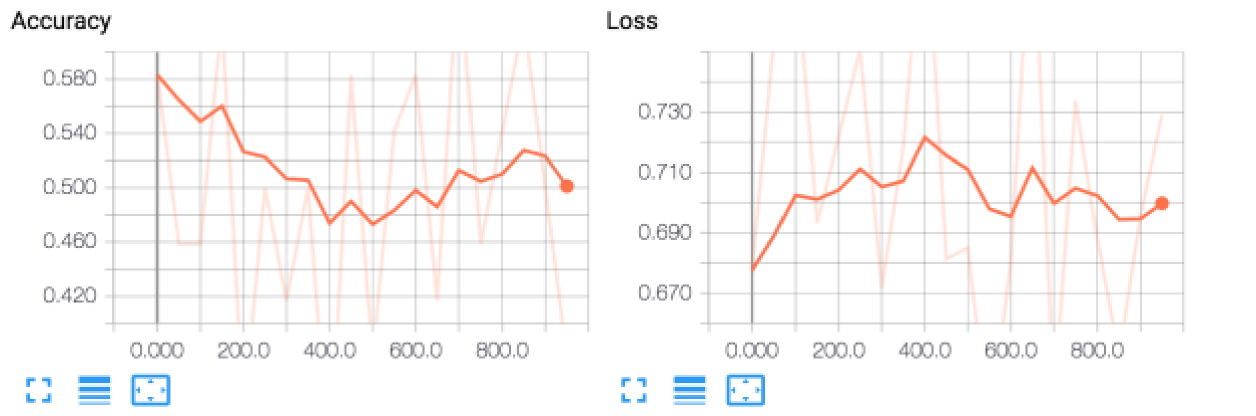

Fig 3 : Accuracy Loss graph for 1,000 iteration

Iteration = 10,000

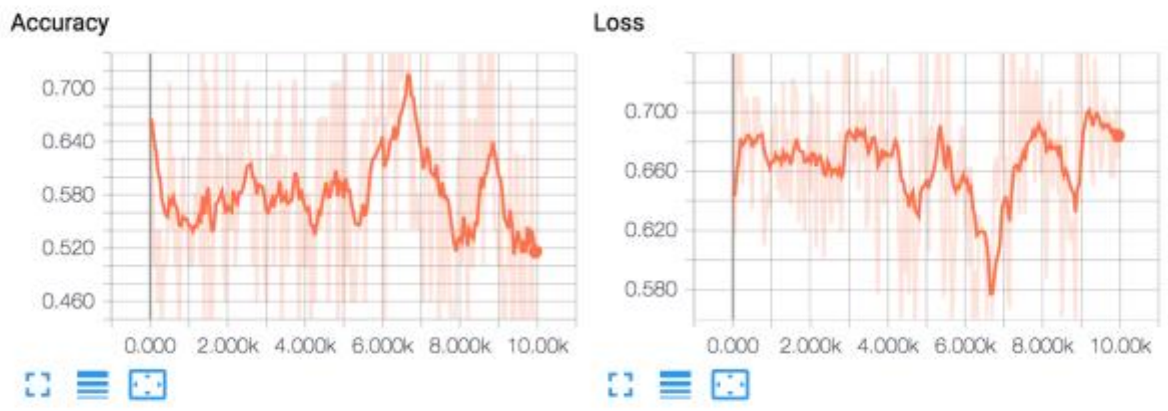

Fig 4: Accuracy Loss graph for 10,000 iteration

Iteration = 100,000

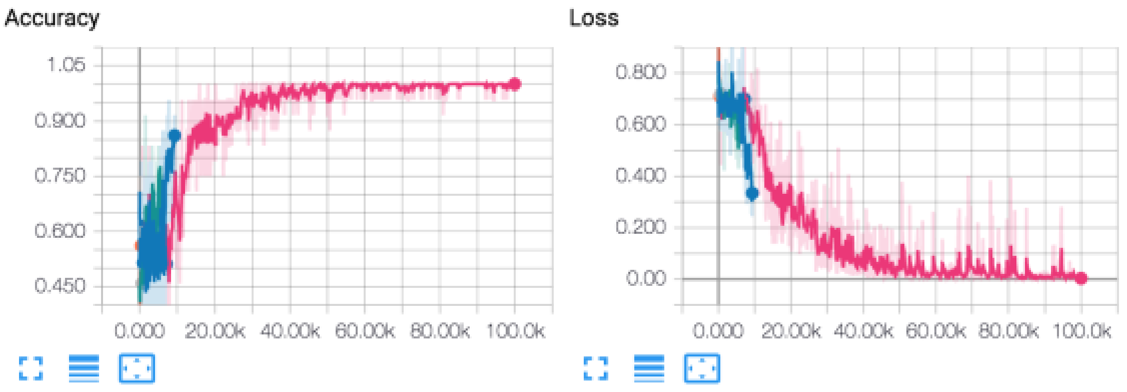

Fig 5: Accuracy Loss graph for 1,00,000 iterations

As we can see that the more we train the model, the accuracy graph converges towards 1 while the loss graphs converges towards 0. These trained model

Testing Graphs

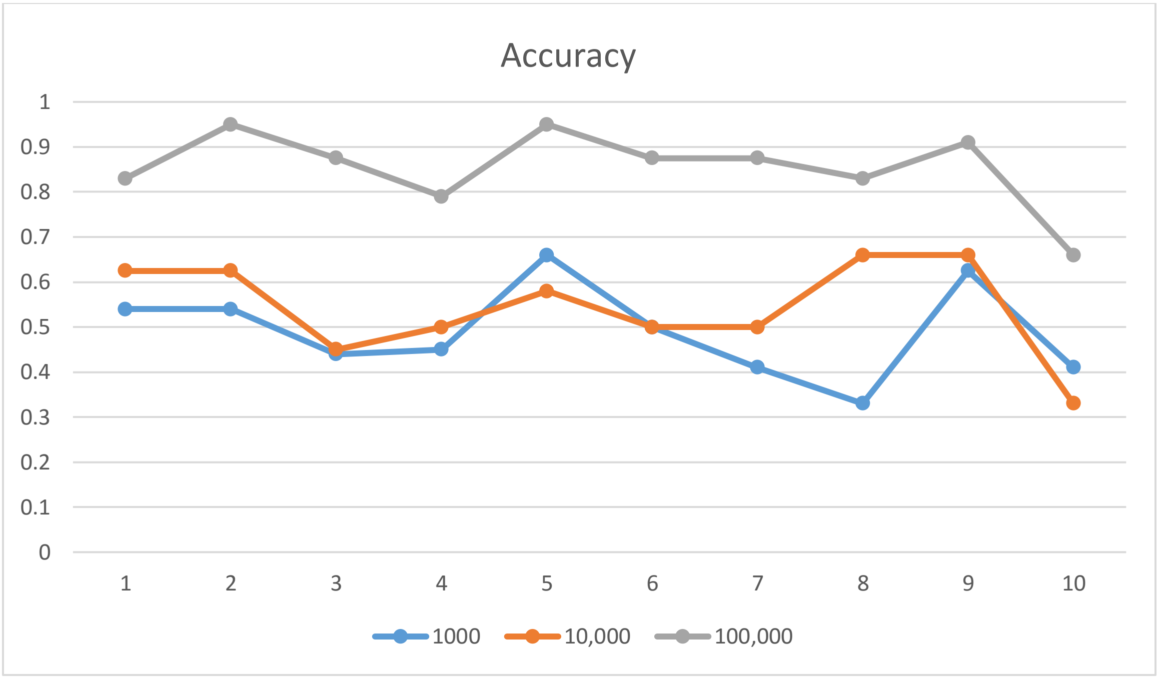

Fig 6: Batch-wise Accuracy graphs with respective iteration.

Here x axis represents batch number, y axis represents the accuracy values. The values are tested while doing 1000, 10,000 and 100,000 iterations. Results may vary while doing testing, due to random selection of graph values.

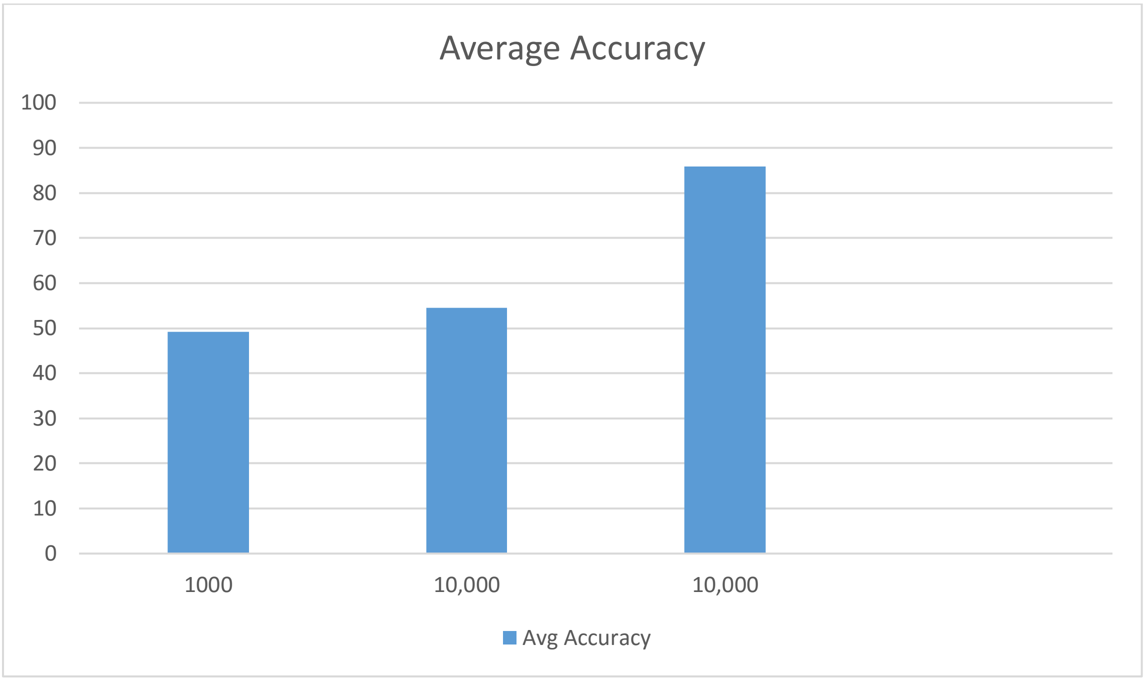

Fig 7: Average Accuracy of respective iteration

These are the average accuracy of the above graph of all the 10 batches. This clearly signifies that more we train the model, the better will be testing accuracy. We also avoid overfitting the model using dropout method.

<u>Conclusion</u>

From the above results, we conclude the linear behavior of training and testing accuracy with overall number of iteration. Use of LSTMs in RNN while tuning the parameters such as Batch size, no of LSTMs units keeping in mind computational complexity of the model given us interesting results and opened the doors in the field of Natural Language Processing.

Future Work

Exploring more sequence models such as Deep Recursive Neural Network and use of bidirectional LSTMs. Also, we can test these models on interesting datasets such as sarcasm detection in debate speeches.

REFERENCES:

[1] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. CoRR, abs/1503.00075, 2015.

[2] https://www.oreilly.com/learning/perform-sentiment-analysis-with-lstms-using-tensorflow

[3] https://github.com/adeshpande3/LSTM-Sentiment-Analysis

[4] https://en.wikipedia.org/wiki/Long_short-term_memory#Peephole_LSTM

[5] James Hong, Michael Fang, Sentiment Analysis with Deeply Learned Distributed Representations of Variable Length Texts