Report HW1

In this homework, I have implemented a bag of words implementation for classification of text as positive (1) or negative (0). Part 1 of the implementation is done using Logistic Regression, where the bag of words counts only include lower case letters, with apostrophe included (‘), and hence removing multiple noisy tokens. It starts with parsing sentence into sparse vectors, and developing matrix based on it, and then training the weight vector and bias which has been initialized to 0. Then, performing multiple epochs and updating weights and bias to get final weights and bias. The training accuracy, precision, and recall are all approximately 0.99. The performance of the trained model on dev data is much different from test data, which is expected. So, dev accuracy is 0.78, precision is 0.76, and recall being 0.81. The performance varies based on the learning rate, based on trial and error I have concluded that best learning rate is 0.1, which also decay each epoch by 0.9, and hence eventually slowly stabilizing the model.

The exploration I did was with the learning rate, I experimented with different learning rates from 0.5, 0.1, 0.01, and 0.001. The plot for average error rate vs no of epochs has been attached in this report, where they are two graphs, one where there is no decay, and other where learning rate decays by 0.9 each epoch. From the plot, it can be understood that the error rate for the learning rate is the best in both case with decay and without decay for learning rate with 0.5. It can also be noticed that the decay is outperforming the learning rate without by a decent margin for each case. But, when the accuracy of dev is better on learning rate 0.1 with decay, even though during training 0.5 has less error, this is most likely because of over fitting the training data with high learning rate.

For the second exploration, I implemented Bigrams and for better features, it includes n-grams, where n is defaulted to 3. The Bigrams, include just bigrams and not unigrams for the sake of just testing out bigram features. The performance of bigram feature on test data is almost 1 for all accuracy, recall, and precision. The performance of bigram feature on dev data is a bad as compared to unigram achieving 0.73 accuracy, most likely because of lack of unigrams and bigrams just overfits the model alone. So, the trigrams includes all unigrams, bigrams, and trigrams performance better than both unigram and bigram achieving dev accuracy of 0.785. After this, we see a decrease in performance for 4-grams and 5-grams achieving accuracy of 0.778 and 0.775 on the dev set, and this trend of decreased performance continues for n grams.

For Deep Averaging Network, I first started by establishing a network using a sequential, and started with 3 layers, with each layer’s depth being 300 to 150 to 75 to 2, the performance of this model was around 0.7, as there is two much depth at the end and it goes from 75 to 2, which is a big jump, and so leading to bad accuracy which also taking

about a minute to train. I switched to 2 layers, and getting accuracy reduced to 0.7. So, final implementation includes 3 layered networks with 300 to 20 to 8 to 2, and with learning rate being 0.01 with 100 epochs, achieving accuracy of 0.78.

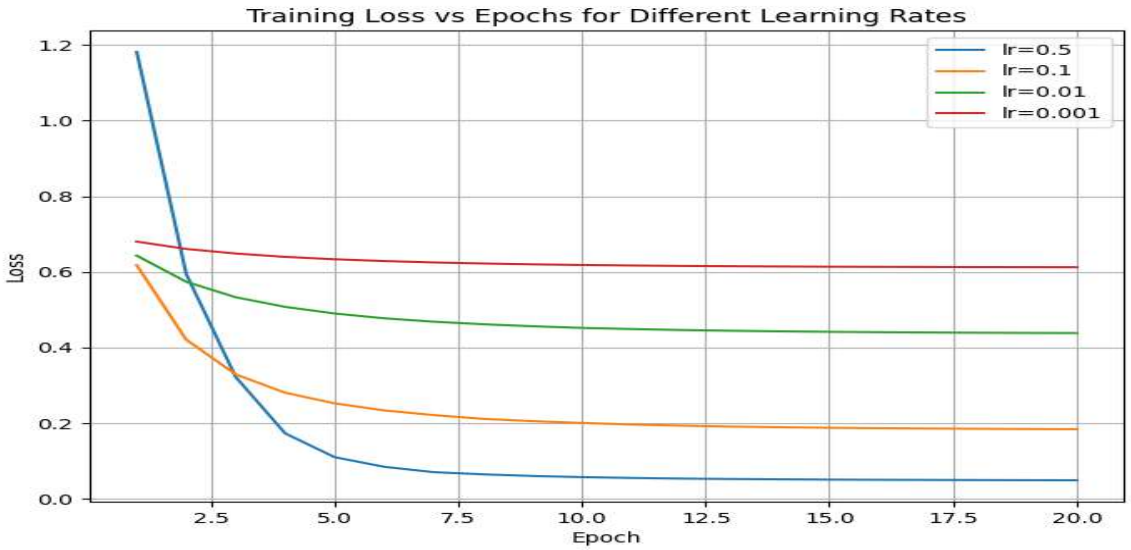

Figure 1 Average Loss Vs No of Epochs without decay

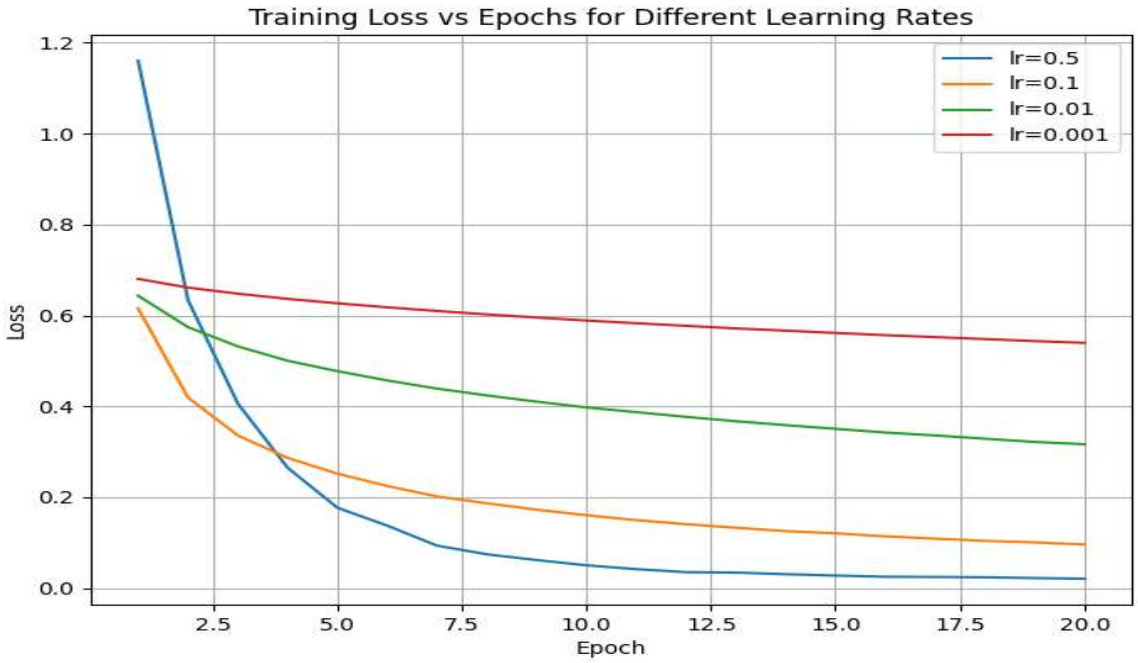

Figure 2 Average Error vs No of Epochs with decay