CMPS242 Homework #5 – Neural Network Tweet Classification

Benjamin Sherman

&

Zayd Hammoudeh

Team Name: "Who Farted?"

1 Homework Objective

Develop a long short term memory-based (LSTM) neural network that classifies tweets as being sent by either Hillary Clinton (@HillaryClinton) or Donald Trump (@realDonald Trump).

2 Dataset Overview

The dataset is a comma-separated list of tweets from either @HillaryClinton or @realDonaldTrump. No additional information is provided by the tweet's text and potentially the class label. The training set consists of 4,743 total tweets while the test set had 1,701 unlabeled tweets.

3 Preprocessor

The dataset consists of tweet

4 Classifier Structure

The LSTM classifier for this homework is required to at minimum the structure shown in Figure 1. There are four primary components, namely the embedding matrix, one-hot vector, long short-term memory network, and feed-forward network. These modules are described in the following subsections.

4.1 One-Hot Vector Input

4.2 Embedding Matrix

4.3 Long Short-Term Memory Network Structure

4.4 Feed-Forward Neural Network Structure

The final structure of our feed forward network is shown in Figure 2. The number of input nodes is dictated by the number of rows in the embedding matrix; as mentioned in Section 4.2, we used a rank of $d = 25$ in our experiments. Our sole hidden layer has $m = 256$ fully-connected neurons. There are two output nodes (e.g., one for "Donald Trump" and the other for "Hillary Clinton). A softmax function normalizes the output probabilities to ensure they sum to probability 1. We selected this paradigm as it simplified the Tensor Flow implementation without affecting the quality of the results.

Our final feed-forward network design used the rectified linear and sigmoid activation functions for the hidden and output layers respectively. Each neuron in the network also had its own bias input.

4.5 Implementation

As specified in the homework description, our network is written in Python (specifically version 3.5.3) using Google's TensorFlow library. Additional packages we used include: Pandas, NumPy, Scikit-Learn (for text preprocessing and vectorizing), and mlxtend ("Machine Learning Library Extension" for generating the one-hot).

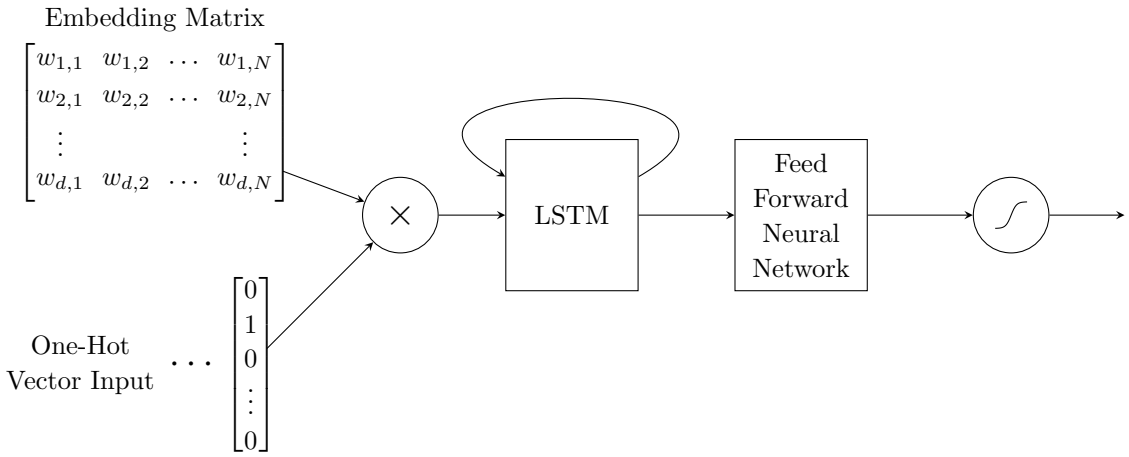

Figure 1: Base LSTM Neural Network Classifier

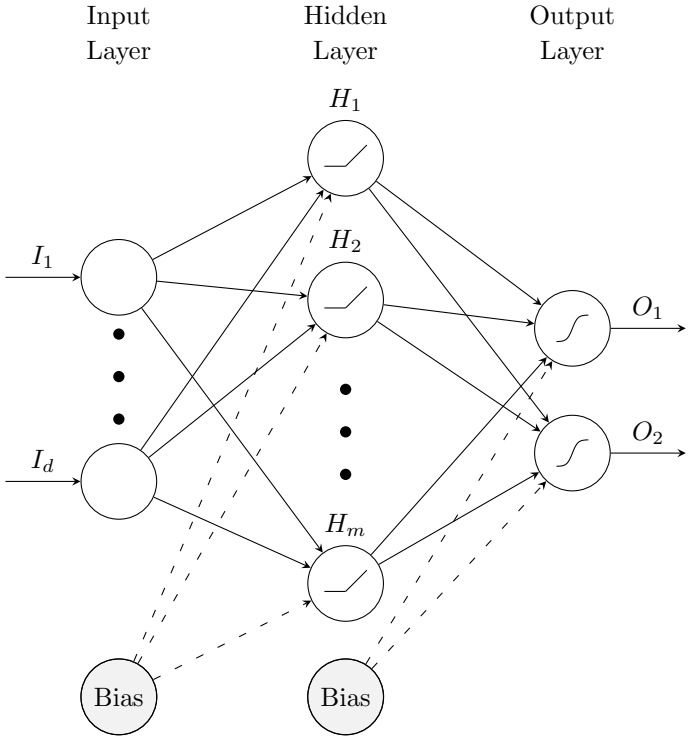

Figure 2: Base Structure of Our Feed-Forward Network

5 Experimental Results

6 Extra Credit #1: Bag of Words Model

In the "bag of words" model, each textual input, i.e., tweet, is transformed into an unordered set of words. Hence, the contextual and word ordering information is discarded. This approach removes any sequential relation in the data; hence, the LSTM added no specific value for training. Hence, we removed the LSTM when performing this experiment and instead trained with just the embedding matrix and the feed-forward network.

Using the previously described neural-network structure, we were able to get 100% accuracy on the complete training set. Likewise, we get an best-case test set of **0.20070** using this approach.

7 Extra Credit #2: Neural Network Experiments

Below are additional experiments we tried beyond the base requirements.

7.1 Extra Credit #2a: Hidden Layer Activation Functions

We experimented with three activation configurations for the hidden layer. In addition to rectified linear, we also tried a "pass-through" activation where the neuron's output was simply $\mathbf{w} \cdot \mathbf{x} + b$, where $\mathbf{w}$ is the weight vector, $\mathbf{x}$ the input, and b the bias term. We also tried to use a sigmoid function. However, these two other activation functions took more epochs to converge (an increase of approximately 200 to 1,000 training rounds) and yield worse testing error.

7.2 Extra Credit #2b: Additional Feed-Forward Hidden Layers

7.3 Extra Credit #2b: Additional Neurons in the Hidden Layer

This is a general (but not strict) correlation between the number of neurons in the hidden layer and the complexity of the function the network can learn. Additional neurons also increase the possibility of overfitting. We experiment with three different quantities of hidden layer neurons, namely 128, 256, and 512. We observed that 128 and 512 neurons had similarly poor performance of approximately **0.24** on the test set. In addition, 512 neurons significantly increased the training time (by a factor of two times). In contrast, 256 hidden layer neurons had a training error of ~ 0.20, that is why we selected $d = 256$ as discussed in Section 4.4.