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 Tweet Text Preprocessor

Since an LSTM is a recursive neural network, it can rely on both vocabulary and contextual information to perform classification. Extensive text preprocessing has the potential to destroy subtle textual information. As such, we performed very minimal text manipulation beyond vectorization. For example, we left all stop words in place. We removed some of the punctuation but left hashtags (#) and exclamation points as we theorized that they may be part of a specific user's unique tweet "signature." This results in a vocabulary of approximately eleven thousand words; it should be noted that a small percentage of this vocabulary is just largely gibberish shortened URLs. We believe that with additional effort, we may be able to derive useful information from these URLs, but we leave that as a future exercise.

4 Classifier Structure

At a minimum, this homework's LSTM classifier must have 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 Implementation Overview

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).

4.2 One-Hot Vector Input

Neural networks perform best when information is encoded in a fixed-length easily decodable fashion. Humans encode words through an ordered series of letters. While this form is compact and well-suited for human consumption, it is not ideal for a machine input. Instead, it is much simpler for the learning algorithm if a word is represented as an N-dimensional bit vector with each of the bits representing one word in the vocabulary. Using this notation, each word would be represented by a vector of $N-1$ zeros and a single bit of 1; this is the reason that this encoding scheme is known as "one-hot" vectors.

A sentence of v words is represented an ordered sequence of v one-hot vectors (with some vectors potentially repeated if a word is repeated in the sentence). In our architecture, each of these one-hot vectors is multiplied by the embedding matrix before being fed into the neural network.

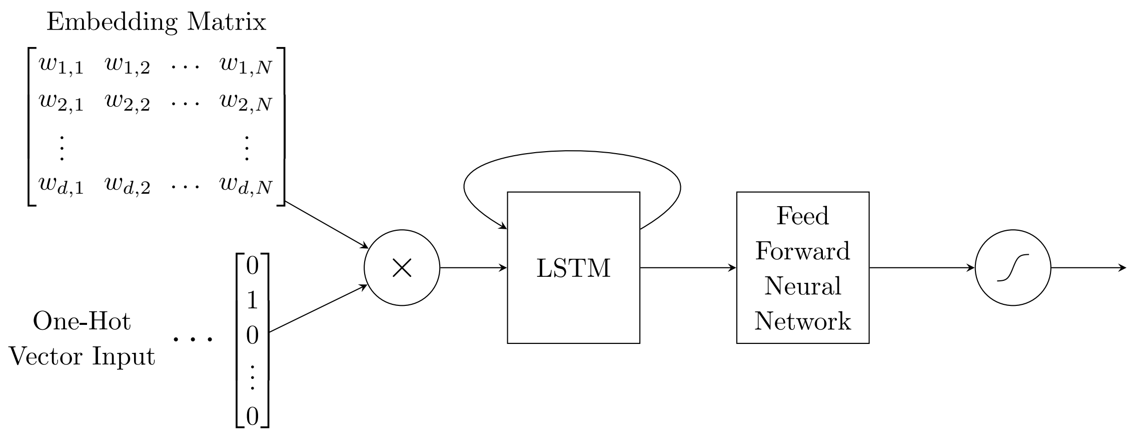
Embedding Matrix
$w_{1,1}$ $w_{1,2}$ $\cdots$ $w_{1,N}$
$w_{2,1}$ $w_{2,2}$ $\cdots$ $w_{2,N}$
$w_{d,1}$ $w_{d,2}$ $\cdots$ $w_{d,N}$
One-Hot Vector Input
0 1 0 0
$\times$
LSTM
Feed Forward Neural Network

Figure 1: Base LSTM Neural Network Classifier

4.3 Embedding Matrix

As mentioned in Section 3, this dataset's vocabulary size, N, is greater than 11,000 words. Using one's personal computer to train a neural network of that size is impractical and will yield severely degraded results. TensorFlow includes built-in support from **embedding matrices**, which via matrix multiplication maps the original sparse N-dimensional space into a much smaller and denser d dimensional space. (In our experiments, $d = 25$ as proposed by Ehsan in lecture.)

The embedding matrix is a learned-object that also can encode relationships between words. For example, in one canonical example, the relationship,

$$\text{Rome} - \text{Italy} = \text{Beijing} - \text{China},$$

can be shown to be learnable using embedded matrices. This shows that embedding matrices can significantly improve a classifier's performance by deducing an inherent "meaning" to words.

4.4 Long Short-Term Memory Network Structure

NEED TO ADD

4.5 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 4.3, 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.

5 Experimental Results

NEED TO ADD

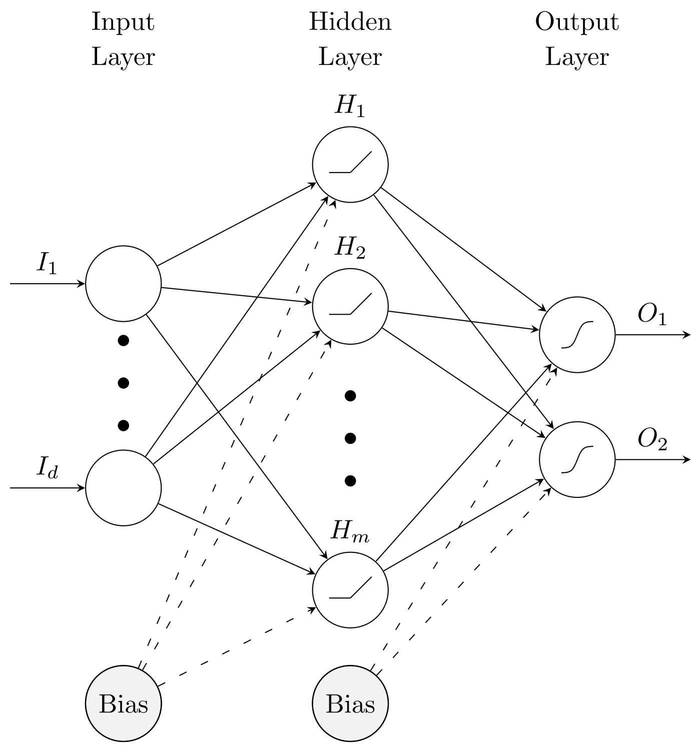

Figure 2: Base Structure of Our Feed-Forward Network

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 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 4.5.

7.3 Extra Credit #2c: Multiple Feed-Forward Hidden Layers

As illustrated in Figure 2, our feed-forward network only had a single-hidden layer. We settled this architecture after experimenting with both two and three hidden layers. Similar to the findings in Section 7.2, increasing the complexity of the feed-forward network by adding more layers had a dual deleterious effect by both increasing the training time and reducing the reported score on the test set.