CMPS242 Homework #6 – Project Proposal

Benjamin Sherman

&

Zayd Hammoudeh

CMPS242 – Project Proposal

1 Primary Project Goal

Implement a character-level recurrent neural network (RNN) that will generate text in the style of President Donald Trump.

2 Training Set Overview

Our plan is to have the learner mimic Donald Trump's oratory style. As such, we will train exclusively on Donald Trump's public speeches. Although some prepared speeches may reflect the speechwriters more than Mr. Trump, we theorize that the president's improvisation digressions are common and unique enough that his style will be plain to the reader.

2.1 Possible Datasets

Multiple datasets currently exist that have collected speeches by Donald Trump. Two of the largest datasets are available as GitHub repositories from users "PedramNavid" and "ryanmcdermott."

2.2 Vocabulary

As with all character-level RNNs, the vocabulary is set of individual characters; it consists of all letters – both capitalized and lowercase, digits (0-9), and punctuation (e.g., comma, space, newline, exclamation point, etc.). It will be dictated by the specific dataset(s) on which we end up training. Preliminary studies indicate that the vocabulary size to be approximately 90 to 100 characters.

3 Neural Network Architecture

The planned structure for our character-level RNN is shown in Figure 1. It consists of five primary stages namely the embedding matrix, LSTM, feed forward network, softmax layer, and the decision engine. We describe each of these stages in the following subsections.

3.1 Embedding Matrix

As mentioned in Section 2.2, we expect the vocabulary to be approximately one hundred characters. As such, some may argue that an embedding matrix is superfluous and may be deleterious. However, we plan to include it primarily for dimensionality reduction. The number of columns in the embedding matrix is still equal to the size of the vocabulary (i.e., number of rows in the one-hot vector). Recall again that this size, $|v|$, is the number of unique characters not the number of different words.

Our team has only two members and given the two to three weeks available to complete this project, time is very much a limiting factor. As the input dimension increases, so does the training time. Our previous experiments indicate that training a character-level RNN without a GPU can take a full day or more even using state-of-the-art hardware. As such, we see the embedding matrix as primarily a acceleration tool.

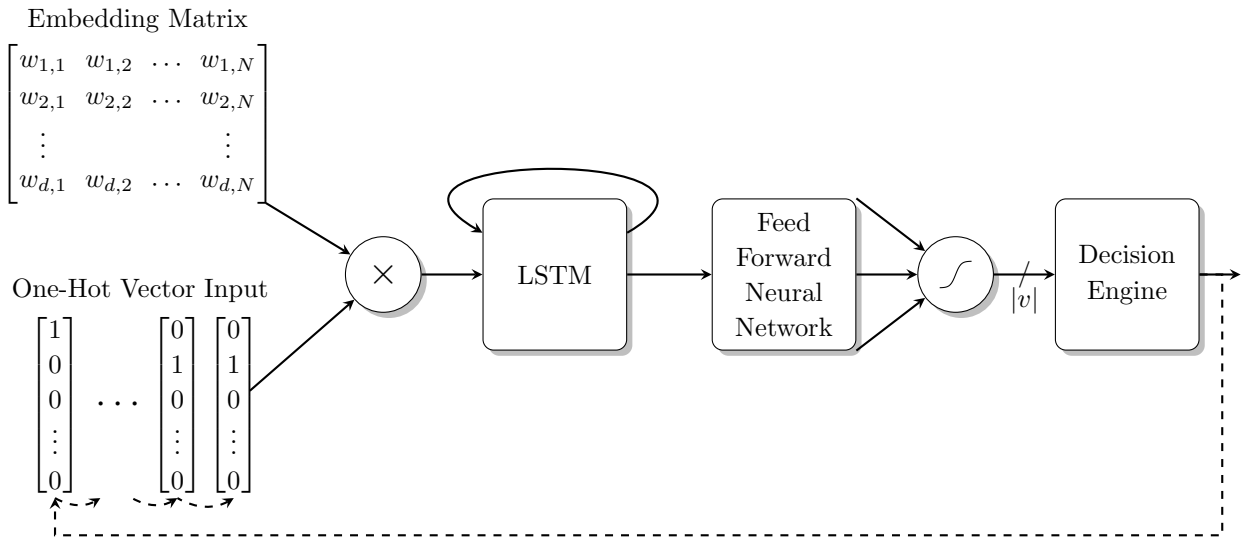

Figure 1: Planned "Trump" Character-Level RNN Architecture

3.2 Softmax Layer

The fourth stage in the learner is the softmax layer. Its role is to normalize the output weights from the feedforward layer.

3.3 Decision Engine

In a typical classifier, the predicted class is the maximizes the output probability. While this approach works well in "one-off" decisions, it is often insufficient when generating a stream of related and inter-dependent characters. What is more, we have observed in our preliminary experiments that always selecting the maximizing character can cause a character-level RNN to enter an infinite loop where it just outputs a single short phrase continuously.

For those reasons, we placed a "decision engine" as the last stage of our learner, as shown in Figure 1. This engine may not always select the character that maximizes the predictor and instead may make weighted randomized decisions. We believe this will reduce the likelihood of the infinite loop generation and will yield more realistic sounding speech. The exact algorithm to be used will be based on our future experiments.

4 Sentence Generation

The character-level RNN has a fixed-length time series window of width, T. The user will enter a string that will seed the RNN. If this seed is shorter than the time series window, the architecture will prepend one or more dummy symbols to the beginning of the string. In contrast, if the string is longer than T, only the last T characters will be used to seed the RNN. This will then be fed into the RNN and a single character will be generated.

Figure 1 shows all subsequent characters are generated (see the dashed lines). Note that the network's output is fed back as the last input when generating the next output. This process continues indefinitely until the output string is of the desired length.