Make Deep Learning Great Again: Character-Level RNN Speech Generation in the Style of Donald Trump

Benjamin Sherman
Department of Computer Science
University of California, Santa Cruz
Santa Cruz, CA 95064
bcsherma@ucsc.edu

Zayd Hammoudeh
Department of Computer Science
University of California, Santa Cruz
Santa Cruz, CA 95064
zayd@ucsc.edu

Abstract

A character-level recurrent neural network (RNN) is a statistical language model capable of producing text that superficially resembles a training corpus. The efficacy of these networks in mimicking Shakespeare, `Linux` source code, and other forms of text have already been demonstrated. In this paper, we show that character-level RNNs are capable of very believably mimicking the language of President Donald J. Trump after training on a corpus of speech transcriptions. We believe our most significant contributions to the study of character level statistical language models are in sampling methodologies; specifically we propose that introducing dropout during the text-generation phase introduces randomness that leads to more believable text.

1 Introduction

The generation of natural language text involves not only the ability to produce speech, but also the relationship between words. In addition, natural language forms including colloquial, formal, legal, mathematical/scientific, etc. Speech patterns are uniquely individual and is influenced by one's background, motivations, and biases.

Natural language generation (NLG) is an open area of research drawing in corporate heavyweights such as Microsoft with Cortana, Amazon with its Alexa platforms, Apple via its Siri subsidiary, and Google through Android and its "Home" product line. All of these attempts at NLG have focused on creating products that speak in a generic manner with broad customer appeal. By itself, that is an immense task but to an extent sterilizes the speech to be as globally non-offensive as possible.

In this project, we simplify the NLG problem by trying to generate speech in the style of only one person – President Donald J. Trump; our selection of him as our focus is based on a very specific rationale. First, Mr. Trump is a particularly polarizing figure in politics meaning the vast majority of people in the country are very familiar with his style. In addition, President Trump has many cliched refrains that are repeated often such "build the wall," "make America great again," "little rocketman," etc. that he repeats continuously. We expect that this should make it easier for a learner to emulate Mr. Trump's essence. Furthermore, the president tends to use "highly simplistic" words in a "grammatically awkward" fashion. [1] Therefore, if the speech we generate has any defects in structure, grammatical or otherwise, or stuck to infantile language, we expect the reader may be willing to overlook our generators as a "Trump being Trump" tangent especially in a place where he is nearly unanimously reviled like Santa Cruz.

[2] showed that character-level recurrent neural networks have shown surprising effectiveness at generating both natural language and structured text outputs. His work was done in the Lua program-

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

ming language. In this project, we attempt to verify his findings with our Trump character-level RNN written in TensorFlow. We provide more background on character-level RNNs in the next section.

1.1 Character Level RNNs

A character level RNN is a statistical model of language in the sense that it views the behavior of language probabilistically. To better understand this, allow that my brain uses a statistical language model and that my job is to try to finish all of your sentences. You say "I am going to the grocery-". I would guess with high probability that you are about to say "store", though I have to accept that there is a non-zero chance you will say "outlet"- or, perhaps you will stub your toe and say "ouch!" in the middle of your sentence. What a character level RNN does is really no different, except the inferences it learns to make happen on the character level.

Let $\mathcal{C} : V^L \rightarrow P$ be a character-level recurrent neural network, such that V is a vocabulary of approximately 100 characters, L is an arbitrary sequence length, and P is the set of all probability distributions over V. More plainly, we can think of a character-level RNN as a function that takes a sequence of L characters and gives us a probability distribution; in particular, it gives the probability of the next character given the previous L characters. As an example of how the network will ideally behave, imagine that you let $p = \mathcal{C}(\texttt{(M,a,k,e, ,A,m,e,r,i,c)})$. If the network is well trained you should find that p predicts `a` as the next character with high probability.

In order to generate meaningful text with a character level RNN you need to give it a seed, similar to the sentence-to-be-finished from the game described earlier. Prompted with the sequence "We will build a g", the RNN will produce a distribution over the vocabulary. We can sample from this distribution and add the resulting character to the sequence. If the sampled character was 'r', as we would hope, then we now have the sequence "We will build a gr". We can continue this process for arbitrarily many steps. It is important to note that we can not make networks that accept arbitrarily long sequences as input, so at some point we have to start removing a character from the beginning of the sequence for every character we add to the end of the sequence.

1.2 Impracticality of Word-Level RNNs

It is obvious that generating paragraphs of text one character at a time may yield suboptimal results. As part of the feedback to our initial project proposal, the grader specifically asked why we chose to do a character-level RNN instead of a word-level RNN. Superficially, a word-level RNN superficially has clear advantages including that it is less likely to produce spelling mistakes and also makes decisions at a coarser granularity which may yield superior results. When upon a more detail analysis, it is clear that word-level RNNs are impractical.

First, the current Oxford English dictionary has over 170,000 words. A world-level learner would need a separate output node for each of these words. Such a large output is likely to suffer from underflow and floating point errors that would severely degrade the quality of its predictions. In addition, training such a large network would be prohibitively long and would extend significantly past the short duration for this project. Even Google with its near limitless computation and human resources does not do word-level prediction and instead uses morphemes.

One team in the CMPS242 class chose to use a word-level learner. To address the output-layer size issue mentioned previously, this team reduced the vocabulary to several hundred words. They also only generated phrases of approximately 10 words. Such extreme constraints yield results that significantly underachieve character-level RNNs.

2 Training

2.1 Dataset

Character-level RNNs can be provided any user-supplied text as a seed. While some words are substantially more common than others, it does not change the fact that a character-level RNN must be able to generate meaningful outputs from countless many input seeds. Therefore, to achieve acceptable performance, the training set needs to be especially large.

Although President Trump is credited with being the lead author of over a dozen books [3–15], we also deliberately chose to exclude them from the training set also for two primary reasons. First, many of the books featured co-authors or were entirely ghostwritten [16]. As such, it would be difficult to distinguish Trump's own style from those of his writing partners. In addition, most of Trump's books date back to the late 1980's through the early 2000's. Most of the students in the course had not even been born by the time some of these books were written. Hence, the generated text they may yield may not be meaningful to the class's relatively young audience.

Another possible source of training content are Trump's tweets, but we did not use them in this project for multiple reasons. First, Twitter limits tweets to only 144 characters. As such, tweeters deliberately prune content and commentary to fit within this strict type limit. This leads to extensive use of abbreviations, skipping of words, or hastags (e.g., "#MAGA" for "Make America Great Again") many of which are exclusive to the Twitter platform. For example, one of President Trump's signature Twitter mannerisms is to end a tweet with "Sad!"; however, this is not done in everyday speech even by the president himself. In addition, similar to at least some of Mr. Trump's books, many of his tweets are ghostwritten. It has been reported that at least White House social media director Dan Scavino Jr. [17] and Trump lawyer, John Dowd [18], have authored tweets in Trump's name. These "imposter" tweets risk polluting the data set with non-Trump content.

Given the deficiencies associated with training on tweets or the president's book, we decided to exclusively limit ourselves to public speeches made by Mr. Trump since he announced his bid to run for president on June 16, 2015. Some of these public speeches had already been compiled in repository in [19]. Another repository of largely different speeches had been collected in a separate repository [20]. Unlike the first set of speeches which was a static collection in text format, the second set used a web scraper to pull tweets from the University of California, Santa Barbara's campaign speech archive [21]. However, the web scraper had significant bugs that was corrupting the speech output. We modified the script using Python's `BeautifulSoup4` package to properly extract the tweets. We then manually merged the two training sets verifying there were no duplicates.

In total, the training set consisted of more than 115 speeches of varying length. There were more than 365,000 words and two million training sequences. We believe the training set size is more than sufficient for a project of this scope, and we are further confident this is shown in the quality of the generated output.

2.2 Vocabulary

As with all character-level RNNs, the vocabulary is set of individual characters that may be received as part of an input sequence or generated as part of the output. Specifically, the vocabulary consists of all letters – both capitalized and lowercase, digits (0-9), and punctuation (e.g., comma, space, newline, exclamation point, etc.).

The set of characters, v, that comprise the vocabulary is dictated by the training data set. In this project, the size of the vocabulary, $|v|$, was 97.

2.3 Learner Architecture

The base neural network architecture we used is similar to the one proposed in homework #5 and is shown in Figure 1; we specifically refer to the training architecture as the "base" since it is a subset of our complete architecture which is described in Section 3.

The training architecture consists of five primary components, namely: the one-hot vector input, embedding matrix, long-short term memory (LSTM) RNN, the feed-forward network, and finally the softmax-output.

2.3.1 One-Hot Vector Input

As the name indicates, a "one-hot" vector is a zero vector with exactly one element in the vector equal to one. Each element in the vector corresponds to a fixed element in the vocabulary. Hence, the size of the vector is $|v|$, i.e., the size of the vocabulary. Each training example consists of an ordered series of one-hot vectors and a single output one-hot vector. The neural network defines a maximum sequence length which is the maximum number of one-hot vectors that can correspond to a single output. For our network, this maximum sequence length was 50.

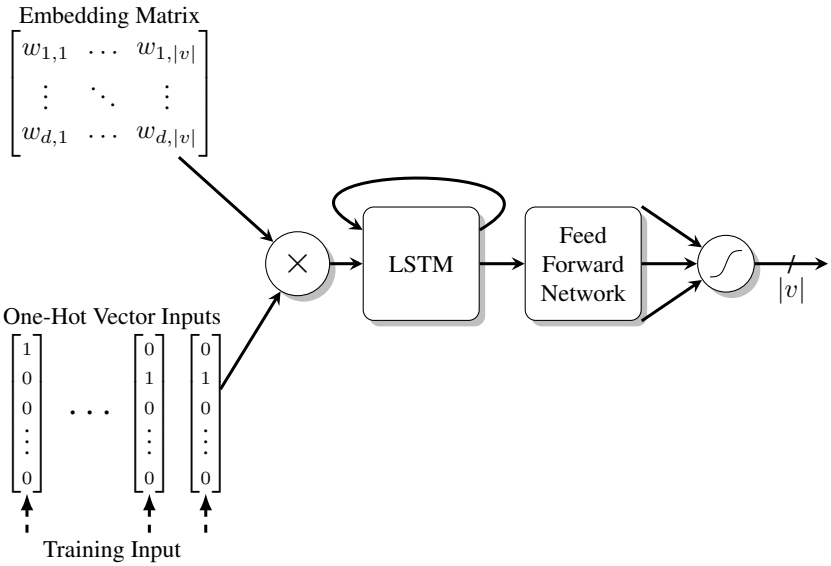

Figure 1: Trump Character RNN Training Architecture

2.3.2 Embedding Matrix

The embedding matrix is a learned object that maps the one-hot vectors from size $|v|$ to size d, which is the number of hidden layers in the LSTM block (see Section 2.3.3). An embedding matrix may seem superfluous for this type of project since the size of the vocabulary is comparatively small. However, we decided to use leave it in our design for several reasons. First, we knew that the duration of the project was relatively short (about two weeks) and our team was smaller than other (only two members). Hence, we expected we may need to accelerate the training reducing the number of neurons in the LSTM. In such a scenario, the embedding matrix would have served as a learned dimensionality-reduction technique.

In addition, it is common for the number of elements in the training set to vary by between 1-5 elements depending on the specific training dataset used. Hence, if a non-Trumpian dataset was to be used for training, it would not be required to train the entire network from scratch. Rather, on the output and embedding matrices would need to be retrained. (Note this feature is not currently supported in our implementation.)

2.3.3 Long Short-Term Memory

2.3.4 Feed-Forward Network

2.3.5 Softmax Layer

A softmax layer is a function, σ, that a real-valued vector, $\mathbf{z}$, of fixed sized, $|v|$, and returns an equal-sized vector, $\mathbf{p}$ whose elements are between 0 and 1 inclusive. Hence, $\sigma : \mathbb{R}^{|v|} \to [0,1]^{|v|}$. The magnitude of each element $z_k \in \mathbf{z}$, is normalized to create value $p_k \in \mathbf{p}$ using the softmax function where:

$$p_k = \frac{\exp(z_k)}{\sum_{j=1}^{|v|} \exp(z_j)}. \tag{1}$$

Therefore, the softmax layer creates a probability vector as the sum of all its elements is necessarily 1. This standardized output is then used by the loss function as described in the next section.

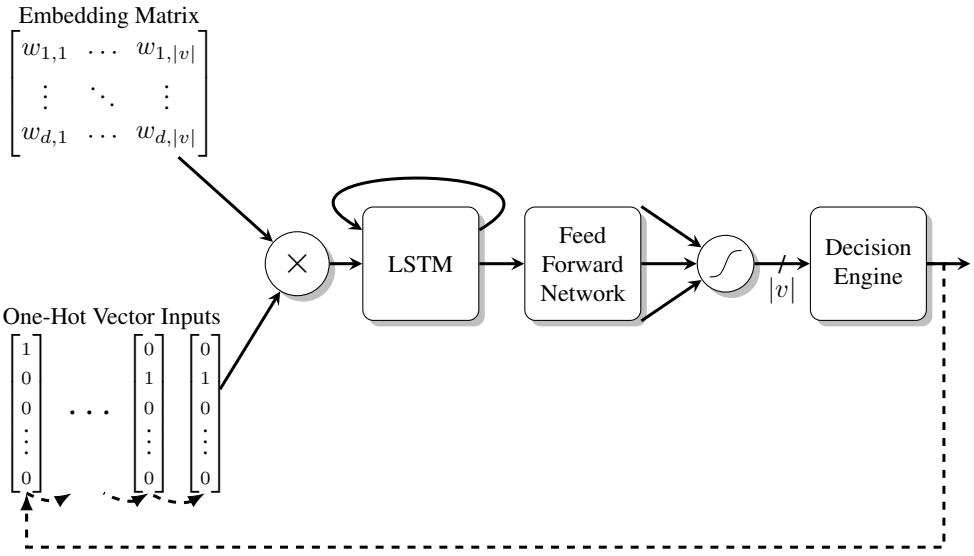

Figure 2: Trump Character RNN Text Generation Architecture

2.4 Loss Function

Our network trainer has two inputs, $X \in (V^L)^n$ and $y \in V^n$. X is an $n \times L$ matrix representation of our sequences, where n is the number of sequences and L is the number of characters in each sequence;] y is a vector of n target values, meaning that y_i is the character following sequence X_i. Let p be an $n \times |V|$ matrix where each row is a probability distribution over $V : p_i(y_i) = 1$.

Our loss per example is the cross-entropy between the distribution given by the network and the true distribution, which is one-hot. Formally, we say that loss, L_i, on example i is: $L_i = H(p_i, \mathcal{C}(X_i))$. Our total loss on inputs X and y is then $\sum_{i=1}^{n} L_i$.

2.5 Batch Size, Learning Rate, and Optimizer

2.6 Variable Sequence Length Training

3 Text Generation Architecture

4 Decision Engine

We refer to the post-training text generation algorithm as the decision engine. The name is apt because the problem of generation is: given a distribution of 'the next character', how do you decide what the next character is going to be? An obvious answer would be "take the most likely choice". What if the distribution you receive is nearly uniform? What if the distribution you receive is not accurate? In this section we will evaluate the merits and drawbacks of greedy, randomized, and mixed decision engines.

4.1 Greedy Sampling

The greedy decision engine is in some sense the most obvious: just take the most likely character. A clear advantage is that you are always making the choice that you are always making the most confident choice possible. An issue is that you are not allowing for spontaneity. Text generated with the greedy decision engine tends to enter infinite loops, which we can escape only with the injection of randomness. For example, when we give our best model the seed *"The media is so dishonest."* and generate with the greedy decision engine, the output is *"They want to stop the people of the world. I want to stop the people of the world. I want to stop the people of the world. I want to stop the..."* etc.

4.2 Random First, Greedy Finish

To get out of infinite loops, we need to make our decision engine stochastic. The obvious way to do this would to sample from the distribution returned to us by the network; in other words, the network is giving us a weighted die, so let's roll it. Let's call this the basic random engine.

The problem with the basic random engine is its randomness. The benefit of the greedy engine is that it always makes a confident choice, but the random engine makes no such guarantee. Every time we roll the die given the sequence "My name is Donald Trum" there is non-zero chance it will come up `'Q'`, and this is a huge problem.

One solution is the random-first greedy-finish engine, or RFGF. The basic idea is: always start a new word with a random choice, but in the middle of a word make greedy choices. In practice this means that you use the basic random engine if and only if the last character generated was whitespace, else you use the greedy engine. This gives you the best of both worlds. It is random enough to avoid infinite looping behavior, yet it does not mangle worlds with absurd characters.

4.3 Top-K First, Greedy Finish

Observe that with the RFGF engine, it is still possible to make absurd random choices. Because softmax can not assign 0 probability to any character, you might randomly add a space or exlamation mark immediately following whitespace. This illustrates that even though we get a distribution over a large vocabulary, we should assign 0 probability to some things. Our approach is to do a top-k operation before we sample. This means that before you sample from the distribution you throw away everything but the k most likely values. You then renormalize this sub-distribution so that the probabilities sum to 1. Now you have a k-sided that die that you can roll. We call this the top-k first greedy finish engine, or TFGF.

4.4 Randomization through LSTM Dropout

5 Conclusions

5.1 Future Work

References

[1] O. Goldhill, "Rhetoric scholars pinpoint why Trump's inarticulate speaking style is so persuasive," *Quartz.com*, Apr 2017.

[2] A. Karapathy, "The unreasonable effectiveness of recurrent neural networks," May 2015.

[3] D. Trump and T. Schwartz, *Trump: The Art of the Deal*. Random House, 1987.

[4] D. Trump and C. Leerhsen, *Trump: Surviving at the Top*. Random House, 1990.

[5] D. Trump and K. Bohner, *Trump: The Art of the Comeback*. Times Books, 1997.

[6] D. Trump and D. Shiflett, *The America We Deserve*. Renaissance Books, 2000.

[7] D. Trump and M. McIver, *Trump: How to Get Rich*. Random House, 2004.

[8] D. Trump, *Crippled America: How to Make America Great Again*. Threshold Editions, 2016.

[9] D. Trump, *Time to Get Tough: Making America Great Again!* Regnery Publishing, 2015.

[10] D. Trump and B. Zanker, *Think Big and Kick Ass: Make it Happen in Business and Life*. Morrow Avon, 2007.

[11] D. Trump and K. Bohner, *Trump: The Art of the Comeback*. Times Books, 1997.

[12] D. Trump and M. McIver, *Trump Never Give Up: How I Turned My Biggest Challenges Into Success*. John Wiley & Sons, 2008.

[13] D. Trump and R. T. Kiyosaki, *Why We Want You to be Rich: Two Men, One Message*. Plata Publishing, 2006.

[14] D. Trump, *Trump: The Best Golf Advice I Ever Received*. Crown Publishers, 2005.

[15] D. Trump and R. Kiyosaki, *Midas Touch: Why Some Entrepreneurs Get Rich, and Why Most Don't*. Plata Publishing, 2012.

[16] J. Mayer, "Donald Trump's ghostwriter tells all," *The New Yorker*, Jul 2016.

[17] A. Ohlheiser, "The (other) man behind the curtain of Trump's twitter account is revealed ... again," *The Washington Post*, Oct 2017.

[18] K. Phillips and A. Blake, "Trump on Michael Flynn's guilty plea: It's a 'shame' because he had 'nothing to hide'," *The Washington Post*, Dec 2017.

[19] R. McDermott, "1mb archive of donald trump speeches," 2017.

[20] P. Navid, "All of Trump's speeches from June 2015 to November 9, 2016," 2017.

[21] "2016 presidential election speeches and remarks," 2016.