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 ability to understand the relationship between words. [1] In addition, natural language takes many very different forms including colloquial, formal, legal, mathematical/scientific, etc. Speech patterns are uniquely individual and are 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. We expect that this repetition should make it easier for a learner to emulate Mr. Trump's speech essence. Furthermore, the president tends to use "highly simplistic" words in a "grammatically awkward" fashion. [2] Therefore, if the speech we generate has any defects in structure – grammatical or otherwise – or if uses infantile language, we expect the audience may attribute these shortcomings to the president himself instead of our tool (especially in a place where Mr. Trump is nearly unanimously derided like Santa Cruz).

[3] demonstrated the surpising effectiveness ofcharacter-level recurrent neural at generating both natural language and structured text. His work was done in the Lua programming language. In this

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

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 that 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, a seed needs to be provided. This represents the sentence or thought to be finished using the game described earlier. Prompted with the sequence "We will build a g", the RNN will produce a distribution over the vocabulary. We then 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 characters. 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 use a character-level RNN instead of making decisions at the word-level. Superficially, a word-level RNN has clear advantages including that it would not produce spelling errors and also that it makes decisions at a coarser granularity, which would be expected to yield superior results. However, upon a more detailed analysis, it is clear that word-level RNNs are impractical.

First, the current Oxford English dictionary has over 170,000 words. [4] A world-level learner would need a separate output node for each of these words (there may be optimizations that could be made to reduce this output count such as using morphemes but that is no longer a word-level RNN). 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 of this project. Even Google with its near limitless computation and human resources does not do word-level prediction.

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 or less. Such extreme constraints yield results that significantly underachieve character-level RNNs.

I was using "v" lower case for the set of characters so if you want to stick with upper case, I will change my sections

I use P for probably so we may want to use a different letter here

I am not sure why you switched to lower case here other than maybe it is not a vector. May want to explicitly define the change

I was unsure why you used single quotes here. I assume there is a reason, so i would be interested to learn.

2 Training

This section describes the techniques we employed to train our Trump character-level RNN. Specifically, it outlines the training dataset, the structure of our base neural network as well as implementation-level details including the optimizer, learning rate, and batch size.

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 [5–17], 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 [18]. 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. [19] and Trump lawyer, John Dowd [20], 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 [21]. Another repository of largely different speeches had been collected in a separate repository [22]. 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 [23]. 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 3.2.

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 layer.

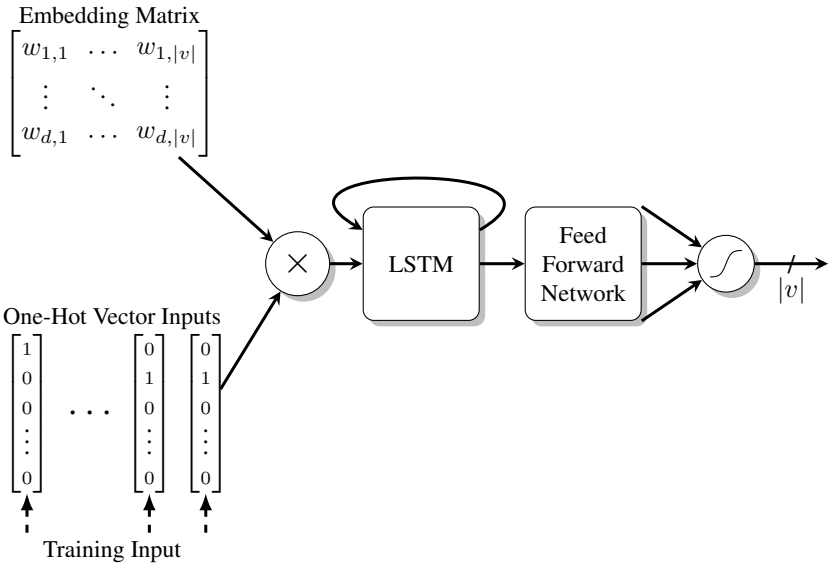

Figure 1: Trump Character-Level RNN Training Architecture

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.

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

Dropout Dropout is a computationally inexpensive technique to provide regularization in a neural network. It mimics the bagging technique used when training an ensemble classifier by temporarily deleting units within the network.[24] TensorFlow supports input, state, and output dropout, which may be used independently or in conjunction with one another. We experimented with different dropout modes and probability. We did not see a significant difference between the settings and settled on output dropout with probability 0.8.

2.3.4 Feed-Forward Network

In the same way that the embedding matrix maps the input to the network to the input of the LSTM, the feed-forward network maps the output of the LSTM to the output of network (i.e., the softmax layer). Most of the power of a character-level RNN comes from its LSTM; hence, to prevent extreme overfitting, we opted for a simple feed-forward output network with only a single hidden layer of 256 fully-connected neurons. We also observed that we achieved superior results if we used the rectified linear activation function for both the hidden and output layers.

2.3.5 Softmax Layer

A softmax layer is a function, σ, that a real-valued vector, $\mathbf{x}$, of fixed sized, $|v|$, and returns an equal-sized vector, $\mathbf{p}$ whose elements are between 0 and 1 inclusive. Hence, $\sigma : \mathbb{R}^{|v|} \rightarrow [0, 1]^{|v|}$. 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:

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

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.

2.4 Loss Function

Our network trainer has two inputs, $X \in \left(V^L\right)^n$ 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 $\sum_{i=1}^{n} L_i$.

It may be better if you define the variables before you use them. I had trouble following this a bit.

2.5 Batch Size, Learning Rate, and Optimizer

Three important factors that can have a significant impact on the training of a character-level RNN are batch size, learning rate, and optimizer. A smaller batch size often leads to a better learner. As such, we set our batch size to only 50 sequences. As such, a single training epoch required approximately 40,000 batches. This meant training a single epoch on a modern high-end CPU took about one hour. With so many batches, selecting a high learning rate would risk later batches wiping out the changes made in early ones. As such, we set the learning rate very low, 0.0005.

In homework #5, we observed that TensorFlow's `AdamOptimizer` (which implements adaptive moment estimation [25]) converged the fastest and generally produced the best results. As such, we used it again here for this project.

We should make sure we are consistent in our notation. I may want to discuss this directly to get on the same page

2.6 Variable Sequence Length Training

Training time and output quality are competing concerns when selecting the sequence length of the learner. A shorter sequence limits the contextual information the learner can use when predicting an output, but longer sequences increase training times and at some point have diminishing returns on the output quality.

In homework #5, each tweet had a fixed, predefined sequence length. In contrast, during speech generation, the learner is provided a user-created seed text that may be shorter or longer than the maximum sequence length, n. The case of a longer sequence is arbitrarily handled by only considering the n most recent characters. In contrast, shorter sequences are more challenging since generally the learner is only trained at the maximum sequence length. To address this, we added *variable sequence length training* where we train our network on multiple sequence lengths between the minimum

It is often good practice to get in the habit of using the tilde as a non-breakable space to decide how symbols will be split across lines.

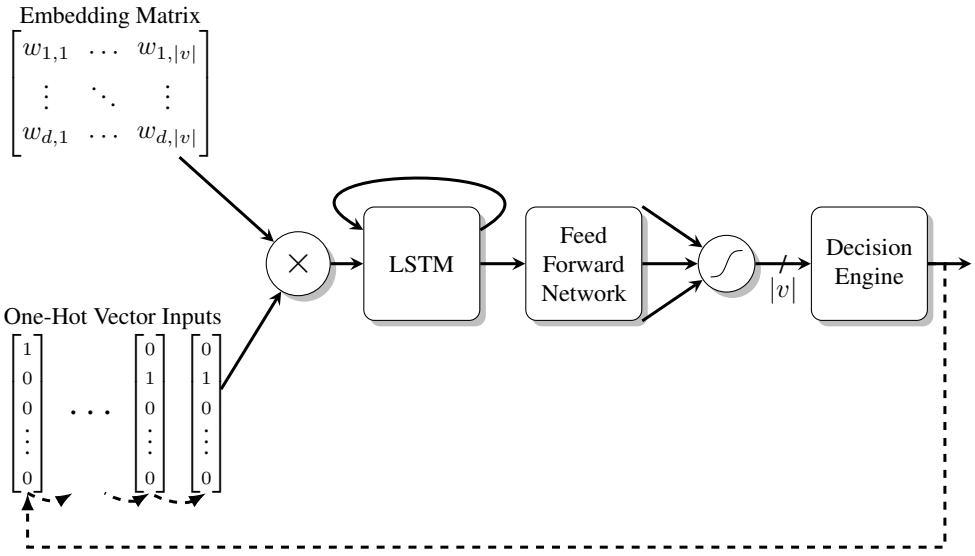

Figure 2: Trump Character-Level RNN Text Generation Architecture

supported (e.g., 10 characters) and n. This is expected to improve our network's performance on shorter sequences.

3 Text Generation

The approach used for text generation is very similar to the training methodology described in Section 2. In this section, we discuss the specific changes made to the architecture to enable text generation.

3.1 Seeding

Rather than generating text from random noise, it produces a much better user experience if the user is allowed to supply the initial *seed* text upon which the learner will build the rest of its output. To ensure that the architecture has sufficient context upon which to begin generating text, we required that the seed string be at least 10 characters long. Text generation continues until the specified number of characters (e.g., 500) have been generated.

3.2 Text Generation Architecture

The neural network in Figure 1 is referred to as the "base architecture" since it is only a subset of the complete system used during text generation. Figure 2 shows the complete text-generation architecture, and there are two primary changes from the previously mentioned base. First, we add an entirely new block to the network, which we refer to as the "decision engine." Its role is to select a single output character from the softmax probability vector of length $|v|$; while this may seem like a trivial task, we explain in more detail the specific challenges of the decision engine in Section 4.

The second change is that each generated output character is fed back into the network to generate the next character. Similarly, the previous sequence of characters is shifted by one with the least recent character removed if the sequence length is longer than the maximum, m (e.g., 50). This approach allows the architecture to generate sequences of characters that could theoretically be infinitely long.

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 advantages and disadvantages 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 (RFGF)

To prevent infinite loops, the decision engine of our RNN needs to be 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 (RFGF) engine. 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 the infinite looping behavior, yet it does not mangle worlds with absurd characters.

4.3 Top-*k* First, Greedy Finish (TFGF)

Because the softmax function does not assign zero probability to any character, there is still a chance that absurd letter choices will be made including putting an exclamation point after a whitespace or inserting a newline mid-sentence. This illustrates that even though we get a distribution over a large vocabulary, we should assign zero probability to some choices. Our solution to address this 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 (we used $k = 5$ in our experiments). The sub-distribution is then renormalized so that the probabilities sum to one. Now you have a k-sided that die that you can roll. We call this the top-k first greedy-finish (TFGF) engine.

4.4 Randomization through LSTM Dropout

As explained in Section 2.3.3, dropout is traditionally a technique used exclusively during training. It is not generally used in a "live" environment producing real outputs. However, dropout is very computationally inexpensive and has been highly optimized in TensorFlow making it especially fast. Likewise, using dropout during text generation eliminates the need for random guessing that is associated with both RFGF and TFGF.

We have observed that the best form of randomization may in fact be dropout *during generation*. We are currently in the process of implementing two new decision engine approaches, namely Dropout First, Greedy Finish, which enables dropout only for the first character after a whitespace as well as Dropout And Greedy Always (DAGA) where we always leave dropout enabled and always make the greedy choice. Preliminary data indicates that the latter approach has execution time very close to that of greedy sampling described in Section 4.1.

5 Conclusions

As demonstrated in class, we successfully implemented a character-level RNN that produces text in the style of President. The generated output was so realistic that it could not be distinguished from a real speech by Mr. Trump. It is important to note that not all text produced by the system is coherent; the example presented in class was specifically selected since it significantly superior to the more general output. It is not uncommon that the generator produces incoherent sentences, meaning improvements can still be made to the system. The next section describes a new approach to character-level RNNs that we believe has the potential to significantly improve output quality.

5.1 Future Work

Our RFGF and TFGF engines provide clear advantages over a straight greedy strategy. However, at their core, RFGF and TFGF are both still greedy. They make point in time decisions about the best character to select using exclusively past information, and once a decision is made, it cannot be undone. Rather than selecting a character based solely on past data, we expect a far better decision could be made if we consider the effect of the current decision on *future* decisions as well.

In Section 1.2, we explained that word-level RNNs are impractical for a multitude of reasons. However, we believe that through character-level decisions, we can achieve near word-level results. For example, at the start of a word, rather than immediately selecting a character, the system could select the top-k characters and complete the resulting k words using our greedy-finish approach. The resulting k words could be examined and the best one selected. This approach reduces the number of possible outputs that must be considered at any given time from more than 170,000 words to just k. Likewise, rather than making word-level decisions, even better results still may be achieve if n-gram decisions were made. For example, the k best words could be constructed at a given point in time. From there, each of their k-best descendants could also be constructed. This process would be repeated until a phrase of length n is built.

One of the primary challenges of this word or n-gram-based approach is quantitatively deciding the "best" selection. Coherent speech is subjective, and given the simplistic and awkward nature of Trump's speech [2], we expect that many objective metrics may perform poorly. One approach proposed by Manfred was to select the one with the highest probability (normalized by word length). This approach appears plausible but requires further study.

We are still implementing this aspect of our learner. The implementation is not complete so we are unable to report the results in this document. However, we expect for this work to be completed in early 2018.

References

[1] R. Mitkov, *The Oxford Handbook of Computational Linguistics*. Oxford University Press, 2009.

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

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

[4] "How many words are there in the English language?."

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[20] 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.

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

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

[23] "2016 presidential election speeches and remarks," 2016.

[24] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, 2014.