Language Models for US Presidential Candidates

CS 229 Final Project Report, Autumn 2016; Category: Natural Language

FNU Budianto
budi71@stanford.edu

Jeff Nainaparampil
jeffnain@stanford.edu

Shruti Murali
shru@stanford.edu

***Abstract*—We train three types of language models (LSTM, GRU, and HMM) on datasets of sentences spoken by 2016 presidential candidates, Hillary Clinton and Donald Trump during presidential and primary debates as well as their twitter posts. We use the language models to perform sentence generation and author classification. Results indicate that the LSTM RNN performs the best in terms of perplexity (37.476) and author classification error (10.39%) of our test set, although GRU comes in a close second. Models trained using debates and tweets result in lower classification error but higher perplexity compared to models trained using only debates data. Further work of fine-tuning hyperparameters and augmenting our dataset might improve the performance of our models.**

I. Introduction

The 2016 Presidential Election proved to be a very contentious landscape, with heated language and attacks from both major parties. A national newspaper notes that Donald Trump has a very distinctive, unfiltered language, different from most other politicians [1]. Computational analyses have also been performed on words and sentiment the candidates have used throughout the campaign trail [2] [3]. However, language models might give us a closer look at the vocabulary and sentence structure used by each of the major candidates, even being able to provide a summarized view of their opinions on certain topics.

Specifically, the input to our language model is the sequence of words of the 2016 primary and presidential debates for both Hillary Clinton and Donald Trump, as well as their Twitter posts. For each word, the language model predicts the next word that comes after it by outputting the probability distribution of the next word. We employ three different types of language models, two recurrent neural networks (RNN) and one hidden Markov model (HMM) on this dataset. The two different types of recurrent neural network we utilize are the Gated Recurrent Unit (GRU) networks and the Long Short-Term Memory (LSTM) networks. For each type of language model, we generate two language models, one model trained from the sentences of Hillary Clinton and the other trained by the sentences of Donald Trump.

We use these language models for two different tasks. First, we perform sentence generation of the two candidates using multinomial sampling from the probability outputs of our models. By introducing starting words to the model, our models are able to predict how candidates finish their sentences, which might give insights on a candidate's viewpoints and their general sentence structure. Second, we perform author classification, where we calculate and compare the likelihood of a sequence of words belonging to each candidate using their respective language models and attribute the words to the candidate with the higher likelihood. For this task, we compare the performance of our language models with the classification performance of two classical machine learning models: Naive Bayes with the Multinomial Event Model and Support Vector Machines with the Gaussian Kernel.

II. Related Work

Several different papers have looked into using language models for a variety of purposes on different corpora. More current papers tend to focus on RNNs, particularly the LSTM and GRU models that we use for this project. For example, one such paper looks at training LSTMs on two large corpora, one in French and one in English (Treebank-3 dataset) [4]. They utilize a similar network topology as ours, with an embedding (which they call a projection layer) that maps to a layer of LSTMs, which is finally mapped to the original word. Their test perplexities vary from 110 to 140.

The LSTM configuration for our project is based on a paper that implemented LSTM with dropout [5]. The overall configuration of our project emulates the hyperparameters laid out in the paper with some key differences that we outline in the "Language Models" section. In particular, we utilize their "Medium Configuration", which has LSTM units of size 650 and a decaying learning rate, starting at 1 and decaying by 1.2 every epoch after the sixth. Including dropout prevents larger networks from overfitting to smaller training sets. The paper reports a test perplexity of 78.4 for their best LSTM language model for the Penn Tree Bank [PTB] dataset.

More recently, more work has been done on fitting LSTMs on very large datasets (on the order of one billions words). One such paper, with two layers of LSTMs and 8200 hidden units for each LSTM, produced dramatically good results, with the state-of-the-art test perplexity of 30.2 [6]. The authors comment that larger datasets allowed them to use different configurations for their networks than what would work for a smaller dataset, such as PTB. Their configuration might not be applicable for our small dataset of the transcripts of Trump and Clinton. Thus, we use different configurations to find the best result.

For the authorship attribution and document classification task in general, a variety of different methods have been utilized. One paper on the topic compares five different machine learning approaches on the common benchmark of identifying the authors of the Federalist Papers [7]. The authors find

that, of the methods they compare, classical machine learning algorithms work really well, with regularized discriminant analysis performing the best. Another paper discusses the features and methods available in a document to identify authors [8]. For our paper, we utilize lexical features of word tokens and a probabilistic method, but future improvements to our classification algorithm might add more features and methods to our classifiers.

Although there have been some recent work on linguistic analysis of the political process [9] [10], a literature search on building language models for Hillary Clinton and Donald Trump show no results. To the best of our knowledge, we are the first to build and utilize language models for the two presidential candidates.

III. Dataset and Features

Our project uses two distinct dataset versions: the v1 dataset are the transcripts from the primary and presidential debates from Clinton and Trump and the v2 dataset is created by adding twitter posts from the two candidates to the v1 dataset. Both debates and tweets data are taken from http://kaggle.com. Tables I, II, and III give an summary of the number of sentences and words in each dataset. Lastly, a supplementary Penn Tree Bank corpus of words (http://www.fit.vutbr.cz/~imikolov/rnnlm/simple-examples.tgz) is used to generate a pretrained embedding.

The transcript data first is broken down by sentences. Each sentence is then converted to lowercase and tokenized into a sequence of words using the NLTK toolkit (http://www.nltk.org) [11]. For the Twitter data, we first remove the emoticons, replace all URLs with a URL-token, and tokenize to sentences and a sequence of words using NLTK's TweetTokenizer. For both the transcripts and Twitter data, an EOS-token is added to the end of each sentence. For both the v1 and v2 datasets, we also create a copy with all punctuations removed to see how the models perform with and without punctuation.

The sentences for each candidate are shuffled and split into 80% for the training set, 5% for validation, and 15% for the test set. In training, we use both Clinton and Trump training data to get the list of words so that both models uses the same vocabulary list. We use a fixed vocabulary size (5500 for v1 and 9000 for v2) and the least frequent words are replaced with unknown-token.

For the RNN and HMM language models, the sequence of words is transformed to a vector of integers, where each element represent the index of the word in the vocabulary list. For Naive Bayes, we construct the word-frequency matrix, where the i^{th} row represents the i^{th} sentence, the j^{th} column represent the j^{th} word in the vocabulary list, and the (i, j)-entry of the matrix represents the number of times word j appears in sentence i. For SVM, we use the same matrix but the (i, j)-entry is replaced with either a 0 or 1.

IV. Language Models

We create a separate language model for each presidential candidate. To do so, we apply two major types of models:

Number of Sentences	v1	v2
Clinton	4110	10182
Trump	5330	12066
Total	9440	22248

Table I
Number of sentences in the dataset.

Number of Words with Punct.	v1	v2
Clinton	86719	163633
Trump	77125	154714
Total	163844	318347

Table II
Number of words in the dataset.

Hidden Markov and Recurrent Neural Network, specifically LSTM and GRU. A language model [12] is a learner that outputs the probability of next word given the words prior, i.e. it predicts:

$$P(w_i|w_1, w_2, ..., w_{i-1})$$

To evaluate the two models, we utilize perplexity. Given a known sequence of N words w_i uttered by a candidate, where $\hat{P}$ is the conditional probability given by our language model, perplexity [13] is defined as:

$$\exp\left(-\frac{1}{N}\sum_{i=1}^{N}\log \hat{P}(w_i|w_1, w_2, ..., w_{i-1})\right)$$

Our RNN language model is depicted in Figure 1, where the RNN cell could be an LSTM or GRU. Due to the small size of our dataset, we use only one RNN layer. Each word is first mapped via a word embedding to a vector of size 650. For both the LSTM and GRU, we apply a dropout function with a keep probability of 0.35, adding regularization to our neural network model. Functionally, dropout either scales up by $1/0.35$ with probability 0.35 or sets to zero each element in the embedding vector. This scaling ensures that the expected value of the sum remains unchanged. The embedding vector is fed into the RNN cell, which outputs a vector of the same size. To convert the output of RNN back to its word id, we

	v1	v2
Number of unique words with Punct.	5848	11555
Vocabulary size	5500	9000

Table III
Number of unique words in the dataset and vocabulary size used.

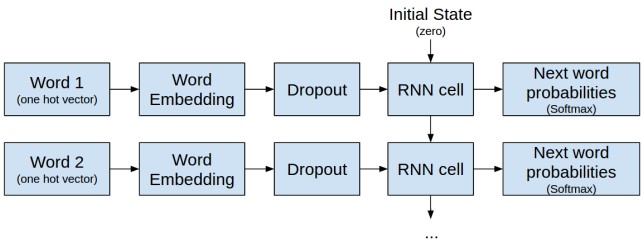

Figure 1. RNN Language Model.

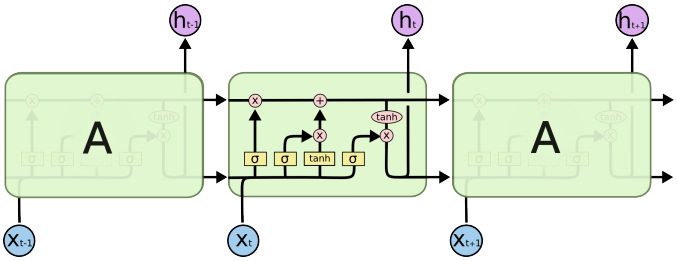

Figure 2. An unrolled LSTM RNN. Source: [15]

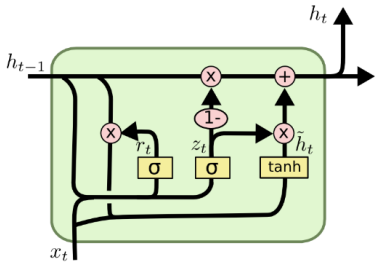

Figure 3. A GRU cell. Source: [15]

multiply the output with a weight matrix and add a bias vector to transform it into a vector the same size as the vocabulary list. We then apply a softmax function to make it a probability distribution.

For the training of the RNN models, the hyperparameters we find that lead to ideal validation perplexity were a learning rate of 1.0 with a decay factor of 0.8, a max gradient norm of 5 (to prevent an exploding gradient), a mini batch size of 20. The training occurs with stochastic gradient descent with log perplexity as the loss function and back-propagation step-size of 35. We then run the training for 18 epochs, where for each epoch we pass through all training data.

A. Long Short Term Memory (LSTM)

Long Short Term Memory [14] cell consists of internal cell state $c_i^{(t)}$, output $h_i^{(t)}$, and input $x_i^{(t)}$, all with the same size (650 for our case). The cell state and output is governed by the following gating functions:

$$
\begin{aligned}
I_i^{(t)} &= W_{I_i} \cdot [x^{(t)}, h^{(t-1)}] + b_{I_i} \\
J_i^{(t)} &= W_{J_i} \cdot [x^{(t)}, h^{(t-1)}] + b_{J_i} \\
F_i^{(t)} &= W_{F_i} \cdot [x^{(t)}, h^{(t-1)}] + b_{F_i} \\
O_i^{(t)} &= W_{O_i} \cdot [x^{(t)}, h^{(t-1)}] + b_{O_i}
\end{aligned}
$$

I, F, O are input, forget, and output gates, respectively. J is new cell state candidate. From these gating functions, the next cell states and outputs are calculated via:

$$
\begin{aligned}
c_i^{(t)} &= c_i^{(t-1)} \sigma(F_i^{(t)}) + \sigma(I_i^{(t)}) \tanh(J_i^{(t)}) \\
h_i^{(t)} &= \tanh(c_i^{(t)}) \sigma(O_i^{(t)})
\end{aligned}
$$

A general diagram of an LSTM cell is given in figure 2.

We utilize an open-source TensorFlow [16] script to implement an LSTM language model, based on [5].

B. Gated Recurrent Unit (GRU)

GRU cells are governed by the following functions [17]:

$$
\begin{aligned}
z_j^{(t)} &= \sigma\left(W_{z_j} \cdot [h^{(t-1)}, x^{(t)}] + b_{z_j}\right) \\
r_j^{(t)} &= \sigma\left(W_{r_j} \cdot [h^{(t-1)}, x^{(t)} + b_{r_j}]\right) \\
\tilde{h}_j^{(t)} &= \tanh(W_{\tilde{h}_j} \cdot [r^{(t)} \cdot h^{(t-1)}, x^{(t)}] + b_{\tilde{h}_j}) \\
h_j^{(t)} &= (1 - z_j^{(t)}) \tilde{h}_j^{(t)} + z_j^{(t)} h_j^{(t-1)}
\end{aligned}
$$

LSTM and GRU have a few differences. First, while LSTM has three gates, GRU only has two gates: read (r_j) and update (z_j). Next, the functions are similar except for the cell state calculation where instead of independent forget and input gates, GRU uses the same update gate to gate both the previous state and the candidate next state. Lastly, the cell state and the output are the same in GRU (h_j). A general diagram of a GRU cell is given in figure 3.

C. Hidden Markov Model (HMM)

A Hidden Markov Model (HMM) [18] is a modeling technique where the system being modeled is assumed to be a Markov process (the future state depends only on the present state) with hidden states. We define the number of hidden states using the part-of-speech (pos) tagger in NLTK toolkit. The number of states can be seen at Table IV. HMM uses three sets of parameters {transition distribution (A), emission distribution (B), initial state distribution (p_0)}, which are initialized empirically:

$$
\begin{aligned}
p_0(s) &= \frac{\sum_{t=1}^{T} 1\{z_t = s\}}{T} \\
A_{ij} &= \frac{\sum_{t=1}^{T-1} 1\{z_t = i \wedge z_{t+1} = j\}}{\sum_{t=1}^{T-1} 1\{z_t = i\}} \\
B_{jk} &= \frac{\sum_{t=1}^{T} 1\{z_t = j \wedge x_t = k\}}{\sum_{t=1}^{T} 1\{z_t = j\}}
\end{aligned}
$$

We combine all the sentences in the training set into one long vector of word indices. Laplacian smoothing is applied on the parameters, and we then use the `hmmtrain` Matlab implementation of the Baum Welch algorithm to train the model. Baum Welch (Forward-Backward) is an EM algorithm where in the E-step, we first run the forward and backward algorithms to compute α_i and β_i:

$$
\begin{aligned}
\alpha_j(t) &= \sum_{i=1}^{|S|} \alpha_i(t-1) A_{ij} B_{jx_t} \\
\beta_j(t) &= \sum_{i=1}^{|S|} \beta_i(t+1) A_{ji} B_{ix_{t+1}}
\end{aligned}
$$

and then set: $\gamma_t(i,j) := \alpha_i(t) A_{ij} B_{jx_t} \beta_j(t+1)$. In the M-step, we re-estimate the maximum likelihood parameters as:

Number of Hidden States	v1	v2
With punct.	42	44
Without punct.	35	35

Table IV
NUMBER OF HIDDEN STATES IN THE HMM.

$$A_{ij} := \frac{\sum_{t=1}^{T} \gamma_t(i,j)}{\sum_{j=1}^{|S|} \sum_{t=1}^{T} \gamma_t(i,j)}$$
$$B_{jk} := \frac{\sum_{i=1}^{|S|} \sum_{t=1}^{T} 1\{x_t = v_k\} \gamma_t(i,j)}{\sum_{j=1}^{|S|} \sum_{t=1}^{T} \gamma_t(i,j)}$$

where T is the length of the sequence and S is the hidden states. Perplexity is calculated using the `hmmdecode` Matlab implementation that computes the likelihood of a sequence using the forward algorithm. We train the HMM models for 500 iterations.

V. Language Model Applications

A. Sentence Generation

Since the RNN language models give us the probability of next word given previous words, we can generate sentences by first feeding starting words (or an EOS-token if no starting words are used) into our model, get the next word, feed the new word back into the model, get another word, and so on until we get EOS-token as the next word, which indicates a full sentence. Instead of picking the word with highest probability as the next word, we do multinomial sampling using the softmax output as the probability distribution, making our sentence generation non-deterministic.

For the HMM, we first run Viterbi algorithm (implemented as `hmmviterbi` in Matlab) to get the most likely hidden state given the starting words (or the EOS-token if no starting words are used). After that, we follow the transition and emission distribution to generate the sentence.

B. Author Classification

For this task, given a sentence s, we can predict the most likely author a using Bayes' Rule:

$$\arg\max_a P(a|s) = \arg\max_a P(s|a)P(a)$$

where $P(s|a) = \prod_{i=1}^{N} P_a(w_i|w_1, w_2, ..., w_{i-1})$, using the chain rule of probability. $P(a)$ is empirically calculated via:

$$P(a) = \frac{\#\text{sentences\_in\_training\_set\_authored\_by\_a}}{\#\text{sentences\_in\_training\_set}}$$

In RNN language models, we calculate $P(s|a)$ by multiplying the probability of the target words taken from the softmax output. In HMM, we calculate $P(s|a)$ using the forward algorithm, which gives us the probability of the sentence (the observed sequence).

We also compare the classification performance of RNN and HMM against the classical machine learning models: Naive Bayes using multinomial event model and SVM with Gaussian kernel:

$$K(x,z) = \exp\left(-\frac{1}{2\tau^2}||x-z||_2^2\right)$$

The Naive Bayes and SVM are implemented similar to the spam classification task in Problem Set #2, but instead of classifying spam or non-spam, we classify Clinton or Trump. Another distinction is that we do not remove the frequent words.

VI. Results and Discussion

We perform a parameter search in order to find the ideal hyperparameters for our model. For more specifics of the parameters we used, please see the "Language Models" section.

The primary metric for our problem is perplexity, as defined above. The comparison of training and test perplexity between the three approaches for dataset v1 (debates only) can be seen at Table V and VI. The comparison of training and test perplexity between the three approaches for dataset v2 (debates + tweets) can be seen at Table VII and VIII. Finally, the comparison of author classification test error can be seen at Table IX, and the confusion matrix of LSTM and dataset v2 with punctuation (which gives the lowest test error) can be seen at Table X. The perplexities of the LSTM and GRU with a pretrained embedding are not tabulated, but generally performed around 26% worse in test perplexity than using a trained embedding for the LSTM in the v1 dataset.

The results indicate that LSTM performs the best in terms of perplexity and and classification error, and LSTM is slightly better than GRU. Both RNN models outperform the HMM. This might be due to RNN models making decisions based on longer histories of words preceding the current word and that the HMM assumption that the hidden states are the part-of-speech tags of the words might be too strong of a bias. Naive Bayes already performs very well in the classification task with only 15.15% error, but the RNN models are able to achieve an even lower error, around 10%. The SVM's high error is most likely because we do not remove the frequent words and because we replace the word-frequency matrix to 0 and 1. Using pre-trained embedding for the GRU and LSTM likely does not give better results, probably because it is trained using a different dataset (Penn Tree Bank corpus) and is not improved during training.

Across all models, perplexity and classification error is better on the dataset with punctuation than on the dataset without punctuation, which indicates that punctuation plays an important role in word sequence prediction. The RNN language models are able to learn the sentence structure and punctuation placement, which can be seen in the generated sentences below. Perplexity is higher on models trained using debates and tweets compared to models trained using only debates data, but the classification error is lower. The higher perplexity is probably due to structural and grammatical differences between sentences in debates and twitter posts.

To get a qualitative analysis of the models, we perform generation of three sentences using the v2 dataset with punctu-

Model	Trump	Clinton	Trump (w/o Punct)	Clinton (w/o Punct)
LSTM	20.063	27.037	28.598	35.349
GRU	19.263	25.128	28.019	33.362
HMM	56.309	80.485	86.330	114.718

Table V
TRAIN PERPLEXITY OF DIFFERENT MODELS FOR DATASET V1.

Model	Trump	Clinton	Trump (w/o Punct)	Clinton (w/o Punct)
LSTM	37.476	53.893	60.724	80.860
GRU	41.304	58.388	68.633	87.740
HMM	98.970	143.695	175.518	229.559

Table VI
TEST PERPLEXITY OF DIFFERENT MODELS FOR DATASET V1.

ation. Each sentence is generated with different starting words: "we will", "if i win", and "i want".

[LSTM] Clinton: "we will do it, together, so that has had a lot of doing in our own country. if i win, in the situation administration, the affordable care act, which is completely part of the very best comment. i want to be a president who thinks joe and security around the world." **Trump**: "we will make america great again! if i win, it 's the best memorial contributions. i want to run against the aliens in this country, will bring back our jobs and make america great again!"

[GRU] Clinton: "we will continue to divert percent of americans and how we 're going to get help guns, you know, look, but it 's always important for people under islam. if i win in the senate, i do n't think it is created. i want to know more about this than anybody else?" **Trump**: "we will be able to bring jobs back to build up to meet the country in the middle east. if i win it, and unlike the republican party, they 're killing us. i want to go to louisiana, ok, a lot of things because whether it is right to march."

[HMM] Clinton: "we will not do the terrorists by our interests is the employees. if i win in ideas, that can n't in to push everything to be laying. i want to do those jobs, he is taken on their women safe." **Trump**: "we will watch out the amazing figure, this picture off companies. if i win that there wants praying soft. i want many big review!"

From the generated sentences, we can see that both RNN models are able to generate sentences that for the most part

Model	Trump	Clinton	Trump (w/o Punct)	Clinton (w/o Punct)
LSTM	29.235	30.923	40.788	41.257
GRU	28.370	30.142	40.115	39.962
HMM	93.243	104.816	136.888	151.323

Table VII
TRAIN PERPLEXITY OF DIFFERENT MODELS FOR DATASET V2.

Model	Trump	Clinton	Trump (w/o Punct)	Clinton (w/o Punct)
LSTM	51.444	56.245	79.275	84.494
GRU	55.833	60.337	85.277	89.426
HMM	151.390	165.186	231.379	242.429

Table VIII
TEST PERPLEXITY OF DIFFERENT MODELS FOR DATASET V2.

Model	With punctuation		w/o punct.	
	v1	v2	v1	v2
LSTM	0.1253	0.1039	0.1279	0.1252
GRU	0.1325	0.1093	0.1403	0.1228
Naive Bayes	0.1436	0.1515	0.1495	0.1755
HMM	0.2898	0.1881	0.3153	0.1833
SVM	0.2931	0.3282	0.3035	0.3591

Table IX
AUTHOR CLASSIFICATION ERROR OF DIFFERENT MODELS.

		predicted author		
		clinton	**trump**	**total**
actual author	**clinton**	1362	166	1528
	trump	181	1630	1811
	total	1543	1796	

Table X
CONFUSION MATRIX FOR LSTM AND DATASET V2 WITH PUNCTUATIONS.

reflect the views the author and make sense grammatically. The RNN models are also able to learn where to put punctuations, resulting in more natural and interesting sentences. The RNN generated sentences tend to outperform the HMM model, which is expected because of the HMM's high test perplexity.

The web UI for sentence generation and author classification using LSTM or GRU with dataset v2 with punctuation is available at http://tiny.cc/cs229-pclm.

VII. CONCLUSION AND FUTURE WORK

Using the debates transcripts and Twitter posts, we are able to build language models that can learn and model the vocabulary and the sentence structure of Hillary Clinton and Donald Trump. We find that the RNN language models outperform the HMM language model due to the ability of RNN to use the knowledge of many previous words. In particular, the LSTM RNN model performs the best. The RNN models also outperformed conventional ML models and the HMM model in the author classification task.

To improve our results in the future, we will continue to add to the Clinton and Trump datasets. As indicated by [6], an order of magnitude larger corpus of spoken words by the two candidates can lead to dramatically improved results. We will also try different types of LSTMs (e.g. LSTM with peephole), as well as perform further hyperparameter (embedding vector size, dropout keep probability, etc.) tuning to see which parameters lead to optimal results. Another approach is to add temperature to the softmax, where higher temperature might lead to more interesting sentences. Trying each of these different parameters via a parameters search can allow us to find models that best generalize to the Clinton-Trump dataset and reduce the test perplexity even further.

Lastly, we also want to improve the web user interface by adding more features, such as sentence generation and classification using HMM.

REFERENCES

[1] L. Mascaro, "'believe me': People say trump's language is affecting political discourse 'bigly'," http://www.latimes.com/politics/la-na-pol-trump-language-20160912-snap-story.html, [Online; accessed 20-November-2016].

[2] K. Murray, "A computational linguistic analysis of the 2016 presidential candidate," http://kentonmurray.com/blogs/comp_linguistic_2016.html, [Online; accessed 20-November-2016].

[3] S. Slobin, "A computer watched the debates. it thought clinton was happy and trump was angry and quite sad," http://qz.com/810092/a-computer-watched-the-debates-and-thought-clinton-happy-trump-angry-sad/, [Online; accessed 20-November-2016].

[4] M. Sundermeyer, R. Schlüter, and H. Ney, "Lstm neural networks for language modeling." in *Interspeech*, 2012, pp. 194–197.

[5] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *CoRR*, vol. abs/1409.2329, 2014. [Online]. Available: http://arxiv.org/abs/1409.2329

[6] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," *Journal of Machine Learning Research*, 2015.

[7] M. L. Jockers and D. M. Witten, "A comparative study of machine learning methods for authorship attribution," *Literary and Linguistic Computing*, p. fqq001, 2010.

[8] E. Stamatatos, "A survey of modern authorship attribution methods," *Journal of the American Society for information Science and Technology*, vol. 60, no. 3, pp. 538–556, 2009.

[9] P. Vijayaraghavan, S. Vosoughi, and D. Roy, "Automatic detection and categorization of election-related tweets," *arXiv preprint arXiv:1605.05150*, 2016.

[10] C. Leuprecht and D. B. Skillicorn, "Incumbency effects in us presidential campaigns: Language patterns matter," *Electoral Studies*, 2016.

[11] E. Loper and S. Bird, "Nltk: The natural language toolkit," in *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1*, ser. ETMTNLP '02. Stroudsburg, PA, USA: Association for Computational Linguistics, 2002, pp. 63–70. [Online]. Available: http://dx.doi.org/10.3115/1118108.1118117

[12] Y. Bengio, "Neural net language models," http://www.scholarpedia.org/article/Neural_net_language_models, 2008, [Online; accessed 12-November-2016].

[13] "Recurrent neural networks," https://www.tensorflow.org/versions/r0.11/tutorials/recurrent/index.html, [Online; accessed 12-November-2016].

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: http://dx.doi.org/10.1162/neco.1997.9.8.1735

[15] C. Olah, "Understanding lstm networks," http://colah.github.io/posts/2015-08-Understanding-LSTMs/, Aug. 2015, [Online; accessed 12-November-2016].

[16] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[17] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," *ArXiv e-prints*, Jun. 2014.

[18] D. Ramage, "Hidden markov models fundamentals," http://cs229.stanford.edu/section/cs229-hmm.pdf, Dec. 2007, [Online; accessed 21-November-2016].