Tweet like @InsertNameHere

Victoria Magdalena Dax
Department of Aeronautics and Astronautics
Stanford University
vmdax@stanford.edu

***Abstract*—In the past month, Twitter saw record levels of election-related conversation on their platform and President Trump's account metrics even hit global records. This work aims at analyzing his tweeting style and attempts to mimic it. We will perform a statistical analysis on his most recent tweets, train a tweet generator as well as a classifier to decipher and predict the semantics of his unique dialect.**

***Index Terms*—Text Generation, LSTM, Naive Bayes', PCA**

I. Introduction

Social media has almost become synonymous with "big data" due to the sheer amount of user-generated content. Mining this rich data can prove unprecedented ways to keep a pulse on opinions, trends, and public sentiment, relevant for marketing, branding, and business as a whole. While there are many popular social media platforms, Twitter provides an interesting blend of data, the tweet contents, and meta-data, such as location, hashtags, users, re-tweets, etc. In the past month, Twitter saw record levels of election-related conversation on their platform and President Trump's account metrics even hit global records. We want to answer the question of why his tweets have so much gravitas, how Donald Trump's unique dialect is so recognizable, and whether we can build a neural network to predict and mimic his vocabulary?

> Despite the constant negative press covfefe
>
> @realDonaldTrump

We will be developing a word-based text/tweet generator using recurrent neural networks (RNNs), perform an analysis of the learned embedding matrix through principal component analysis (PCA) and finally, we will train a Naive Bayes' classifier to distinguish the President's tweets from others and see whether our tweet generator can fool the classifier. We will also be performing statistical word frequency and sentiment analysis to get a better understanding of the data at hand.

Disclaimer: This work is not meant, in any way, to offend or provoke any group. It is not a statement of political opinions, beliefs, or principles of the author.

This project is used for CS229 and CS230. We propose evaluating the tweet generation network towards CS230 and the principal component analysis and Naive Bayes' classifier towards CS229. The code for this project can be found here: https://github.com/victorialena/tweetLikeInsertNameHere.git

II. Related work

Synthetic text generation is challenging and has had limited success in the past. There are two fundamentally different approaches to the task, character-based and word-based models. Both ultimately learn a conditional language model (LM), which gives the probability distribution of the next word in a sequence over a given dictionary of possible words: $p(w_{n+1} \mid w_1, \dots w_{n-1}, w_n)$. Word-based LMs tend to display higher accuracy and lower computational cost than char-based LMs, as they require much bigger hidden layers to successfully model long-term dependencies which means higher computational costs, [1]. But, char-based LMs can mimic grammatically correct sequences, and interestingly enough, model languages with a rich morphology such as Finish, Turkish, or Russian much better then word-based LMs. The bottom line however is that word-based LMs tend to train faster and generate more coherent texts overall.

For a long time, conditional text generation relied on Seq2Seq models, [2], which consists of two Recurrent Neural Networks (RNN), an encoder and a decoder, that when concatenated try to predict the next state sequence from the previous one. However, texts generated with RNNs remain often far from making actual sense, as a single wrong prediction can make the entire sentence meaningless. Another limitation is that it can not be parallelized since RNNs need to process data as a sequence.

A model receives a sequence of tokens, i.e words, and transforms them into static vectors that encode the meaning of the referenced word. These representations are called embeddings and were introduced as Word2Vec model, [3]. These embedding preserve syntactic and semantic word similarities, such that they lend themselves to mathematical operations: KING - MAN + WOMAN = QUEEN.

Recent techniques consists of incorporating context into word embeddings; replacing static vectors with contextualized word representations. ELMo introduced this type of word embedding in 2018, [4], where vectors are learned functions of the internal states of a deep bidirectional language model. With this representation, the model can distinguish between homonyms, such as "rock" referring to a stone or the music

genre.

A new architecture, called Transformers was released by Google in 2017 in the paper "Attention Is All You Need", [5]. Similarly to Seq2Seq, Transformers use an encoder, decoder and a final linear layer. But they consist of self-attention and point-wise, fully connected layers, rather than recurrent or convolutional layers. Consequentially, sequences are not required to be processed in order, which allows for parallelization. BERT and GPT-2, using Transformers in their cores, have shown a great performance in tasks such as text classification, translation, summarization.

III. Dataset and Features

In order to perform our complete analysis of President Trump's tweeting style, we initially estimated requiring around ten thousand tweets. Luckily the president has a tendency to document his every thought on social media. We used Twint, an acronym for Twitter Intelligence Tool - [6], to scrape tweets from specific users. We compiled a dataset of fifteen thousand tweets posted by user @realDonaldTrump. These were used to train the tweet generator with a 5% validation set partition. We further compiled a thousand additional tweets by @BarackObama, @BillGates, @justinbieber, @ArianaGrande, @TheEllenShow, @YouTube, @KimKardashian, @cnnbrk, @twitter, and @TheDailyShow each. These users happened to be the most engaging user of the past year. Their tweets, in addition to 60% randomly sampled ones from the first batch, were used to train the Naive Bayes' classifier with a 10% testing and validation set partition. The following are some excerpt tweets.

> Democrats always liked that position until I took it.
> They hate Fake News and so do I.
>
> @realDonalTrump

Data prepossessing included cleaning out links and web-addresses, filtering out uncommon punctuation and alternate characters - but keeping hashtags # and usertags . We further dropped tweets that were less than three words, only kept the 50 most frequent usertags referenced by the author, and split tweets into individual sentences.

For the tweet generator, the tokenizer normalized everything to lower case and limited the word count to ten thousand. For the Naive Bayes' classifier, we as well normalized the dictionary but kept every word that appeared in at least 5 tweets.

IV. Methods

A. Tweet generation

Conditional language models (LMs) are at the core of text generation. A statistical LM is a probability distribution over sequences of words, such that, given a sequence of length n, it assigns a probability $p(w_1, \ldots, w_n)$ to the whole sequence. In consequence, we can use a conditional LM to find the probability of the next word in a sequence, namely $p(w_{n+1}|w_1, \ldots, w_m)$.

Recurrent Neural Networks (RNN) are dominating complex machine learning problems that involve sequences of inputs. It is a class of neural networks that allow previous outputs to be used as inputs. Such kind of architecture allows the network to learn and generalize across sequences of inputs instead of identifying individual patterns. For each timestep t, the activation $a^{(t)}$ and the output $y^{(t)}$ are expressed as follows

$$a^{(t)} = g_a(W_{aa}a^{(t-1)} + W_{ax}x^{(t)} + b_a)$$
$$y^{(t)} = g_y(W_{ya}a^{(t)} + b_y)$$

where $a^{(t)}$ is passed to the next node as an additional input with $x^{(t+1)}$. The main limitations of standard RNNs is the vanishing/exploding gradient phenomena, induced by the multiplicative nature of gradients that decrease/increase exponentially with respect to the number of layers, which also affects its ability to capture long-term dependencies. Enter Long Short-Term Memory units (LSTM). LSTMs are constituted of internal mechanisms called gates to regulate the flow of information.

First, the previous hidden state and the current input get concatenated and fed into the forget layer. This layer removes non-relevant data. A candidate layer is created using the concatenated state, which holds possible values to add to the cell state. The concatenated state also get's fed into the input layer, which decides what data from the candidate should be added to the new cell state. After computing the forget layer, candidate layer, and the input layer, the cell state is calculated using those vectors and the previous cell state. The output is then computed, and multiplied element-wise with the new cell state to get the new hidden state.

RNNs and LSTMs can also be used as generative models. This means that in addition to being used for predictions they can learn the sequences of a problem and then generate entirely new plausible sequences for the same problem domain. Generative models like these are useful not only to study how well a model has learned a problem, but to learn more about the problem domain itself. For example, by training the tweet generator we find an embedding matrix which encodes the proximity of words in our vocabulary.

Through experimentation, we observed that increasing the number of RNN-type layers beyond three and batch size beyond 512 resulted in overfitting, failing to reach accuracies beyond 0.25 on the validation set. We ended up using two LSTM layers with 256 units each followed by two fully-connected layers, as represented in fig. 1. The input to the network is a tweet subsequence of ten words. For tweets longer than ten words, we adopted a sliding window approach,

splitting the tweet into multiple subsequences of length ten. For shorter tweets, we zero-pre-padded the sequence. The output of the model is a probability distribution of the next word over the vocabulary.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 9, 150)            1500000
_________________________________________________________________
lstm (LSTM)                  (None, 9, 256)            416768
_________________________________________________________________
lstm_1 (LSTM)                (None, 256)               525312
_________________________________________________________________
dense (Dense)                (None, 1250)              321250
_________________________________________________________________
dense_1 (Dense)              (None, 10000)             12510000
=================================================================
Total params: 15,273,330
Trainable params: 15,273,330
Non-trainable params: 0
_________________________________________________________________
```

Fig. 1. Model Summary

A fairly unconventional choice was using a SELU activation for the first dense layer, which helps with vanishing gradients. Since we are mapping a categorical distribution, using a softmax activation on the final layer is a natural choice as well as using the categorical cross entropy loss.

$$\ell_i = \sum^{|V|} -y_i^{(d)} \log \hat{y}_i^{(d)}$$

Using an Adam optimizer with a learning rate of $8e^{-4}$, a batch size of 128 and an embedding size of 150, we trained the model over 100 epochs.

B. Principal Component Analysis

Principal component analysis, short PCA, is a non-probabilistic approach to finding a subspace representation of given data. The aim is to keep as much information as possible, thus the method is to maximize the cumulative variance over all data points when projected onto the subspace. For this project, we wish to use PCA to map the learned word embeddings into 2D space in order to gain some insight into the structure of President Trump's vocabulary.

First, we center and whiten the data, by subtracting the mean and normalizing the rows of our embedding matrix. We compute the empirical covariance matrix and find it's eigenvalues and vectors. Then we choose the eigenvectors corresponding to the two highest eigenvalues - those are our two principal components - and project each row of the embedding matrix onto this newly defined plane. How representative these projections are can be determined by evaluating $\frac{\sum^k \lambda_i}{\sum^d \lambda_i}$. Our 2D representation yields only 6.5%; we would yield 9% if we were to map to 3D space and for 90%, which is commonly considered a significant representation, we would require 100D space.

C. Naive Bayes' classifier

We used a Naive Bayes' approach to classify tweets, distinguishing whether or not user @realDonaldTrump was the author. This approach is part of generative supervised learning, which means that instead of learning $p(y \mid x)$ directly, $p(y)$ and $p(x \mid y)$ are learned and Bayes' rule is applied when making predictions. The assumption this specific approach is making, is that all features x are considered independent given label y. We opted for a multinomial event model, where each feature $x_i^{(j)}$ represents the ith word in the jth tweet of length d_j. Thus $p(x_i \mid y) = \phi_{x_i|y}$ maps the probability of word x_i over the vocabulary given y, which follows a categorical distribution. $p(y = 1) = \phi_y$ follows a binomial distribution, since we attempt a binary classification. Thus the log-likelihood is defined as

$$\ell = \sum^{n} \log p(y_j) + \sum^{n} \sum_{i}^{d_j} \log p(x_j^{(i)} \mid y_j)$$

The update rule can be derived from the maximum likelihood estimates as follows.

$$\phi_{k|y=1} = \frac{\sum_j^n \sum_i^d {}_j 1\{x_j^{(i)} = k, y_j = 1\} + 1}{\sum_j^n 1\{y_j = 1\} + |V|}$$

$$\phi_{y=1} = \frac{\sum_j^n 1\{y_j = 1\}}{n}$$

Evidently, we applied Laplace smoothing to avoid zero estimates for rarer words in the vocabulary. We established our dictionary by evaluating nine thousand tweets by @realDonaldTrump and a total of ten thousand tweets by other popular verified Twitter users as outlined in sec III. Only lower-cased words that appeared in over 5 tweets were added to the dictionary.

V. Experiments/Results/Discussion

We started by evaluating a word frequency analysis; the results are represented in fig 2. It comes to no surprise that the words "people" and "country" are recurrent themes as President Trump holds a public office. We further find the emblematic "great" and "fake news" in his frequent vocabulary, but he surprisingly tweets more often "democrat" than "republican".

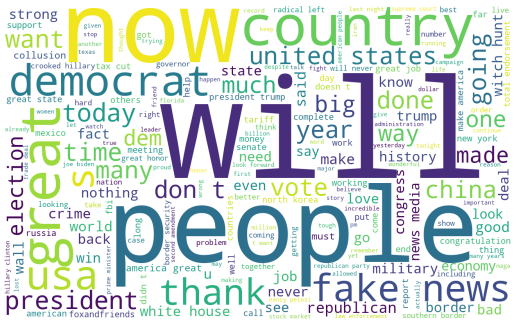

Fig. 2. Word Could of 150 most frequent words

VADER, short for Valence Aware Dictionary and sEntiment Reasoner - [7], is a lexicon and rule-based sentiment analysis tool that is specifically attuned to sentiments expressed in social media. We take the compound score, a metric that calculates the average over all the words' individual ratings, to perform sentiment analysis on tweets by @realDonaldTrump. The results are presented in fig. 3, with -1 being the most negative and +1 the most positive score possible. We can see that the majority of his tweets are very polarizing.

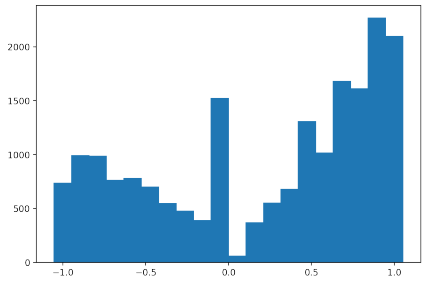

Fig. 3. Sentiment Analysis

The architecture described in sec. IV-A and depicted in fig. 1 was the model that best balanced validation accuracy and coherence of generated tweets. The later metric is evaluated by generating 50 tweets of maximum ten words each and labelling them whether they are a grammatically coherent sequence of words.

Through hold-out cross validation, we performed a hyperparameter grid-search over learning rate, batchsize and embedding size. The grid-search varied the learning rate between $1e^{-3}$ and $1e^{-4}$ in a log-uniform manner, and evaluated batchsizes of 1024, 512, 256, and 128, and embedding sizes of 50, 100, 150 and 300. We found that a learning rate of $8e^{-4}$, a batchsize of 128 and an embedding size of 150 yielded the best results. Lower learning rates resulted in requiring over 250 training epochs while a learning rate of $1e{-}3$, which is the default for the Adam optimizer in Tensorflow, led to a premature stagnation of the loss. The resulting training history is shown in fig 4. It still seems that the model is overfitting to the training data a bit. The metadata was satisfactory, thus we didn't attempt training with more data or trying smaller network architectures.

We found a final validation accuracy of 0.654 and a coherence of 54%. The following are some examples of generate tweets:

> Make America great again and then keep America great!
> I will win again tonight.
> The tax system is paying for the federal reserve.
> Socalled birthright citizenship which costs our country billions of dollar!
>
> @InsertNameHere

Using principal component analysis, our learned embedding

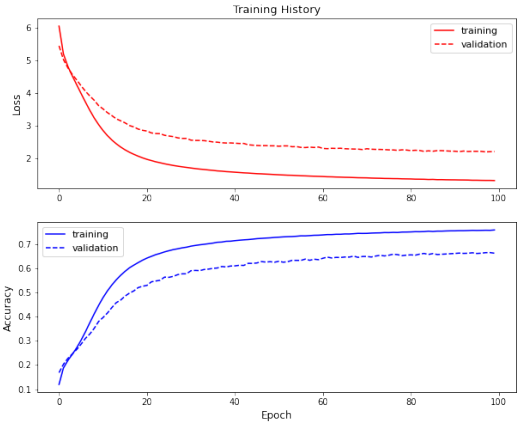

Fig. 4. Training History

matrix is mapped to 2D space, fig 5. Interestingly, "nancy", "joe", "hillary", "bernie" and "donald" are mapped to the upper right corner while "pelosi", "biden", "clinton", "sanders" and "trump" are mapped closely together in the center of the space, splitting first from last names. Surprisingly, there was no coherent mapping of locations, states such as "texas" or "carolina" and countries such as "russia" or "china" seem to be projected uniformly across the plane. Finally, we find vocabulary relating to news, like "press" or "@cnn", to be projected along the $100\,\mathrm{deg}$ axis, and words referring to a public office, such as "taxes" or "party", perpendicularly along the $-115\,\mathrm{deg}$. Words referring to the election, for example "rally" or "ratings", are mapped to the space in between. What a coincidence.

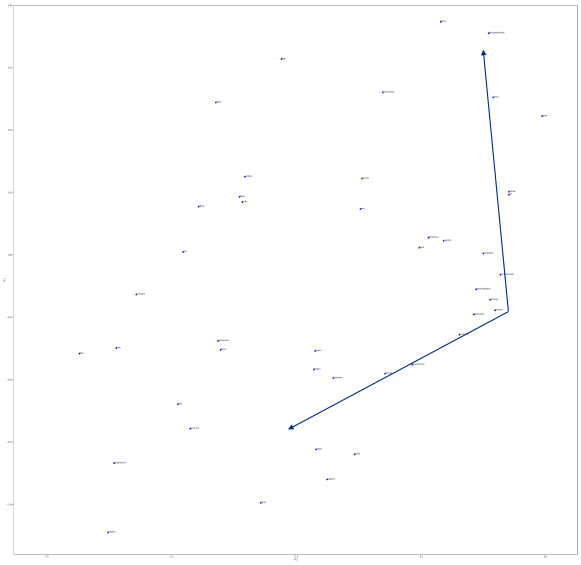

Fig. 5. Word Map Excerpt

The Naive Baye's classifier is trained to distinguish the

President's tweets from tweets by other verified users - namely @BarackObama, @BillGates, @justinbieber, @ArianaGrande, @TheEllenShow, @YouTube, @KimKardashian, @cnnbrk, @twitter, and @TheDailyShow. The aim is to see whether Trumps style is easily distinguishable and whether our tweet generator can fool the classifier. The dictionary is compilied from all lower-cased words that appear in more than 5 tweets. The final dictionary size was 4529 words. We trained the classifier according to the update rule outline in sec IV-C with a 10% validation partition. The resulting accuracy was 0.79%. Table I shows the confusion matrix of the classifier's performance measured on the validation set. We find a precision of 80% and a recall of 73%

	predicted T	predicted F
True	948	340
False	223	1223

TABLE I
CONFUSION MATRIX

Letting the classifier predict the author of our generate tweets, we find it labels 85% of our tweets as President Trump's. A most indicative word analysis further reveals that "respected", "socalled", "corrupt", "ratings", "@seanhannity" "strongly", "badly", and "@foxandfriends" are most characteristic for his tweeting style.

Finally, we devised a Turing test to see whether our friends would be fooled by the tweet generator. We asked 5 people to label 50 tweets each, the results are shown in table II. The precision lies at 58% and the recall is at 43%, suggesting that people can still distinguish between generated and real tweets wit ah higher precision than the classifier. However a significant amount, 35% to be exact, of our generated tweets are labelled as real by humans.

	predicted T	predicted F
True	74	26
False	53	97

TABLE II
TURING TEST

VI. CONCLUSION/FUTURE WORK

We performed a full analysis the tweeting style of Twitter user @realDonaldTrump. First, the statistical word frequency analysis showed as expected the predominant usage of terms related to a public office and a significant usage of phrases people have come to associate with President Trump's tweets.The sentiment analysis further confirmed his polarizing writing style and content. Next, we trained a classifier to distinguish his tweets among other randomly samples ones. The Naive Bayes' classifier had no trouble fitting to words in the vocabulary that were most relevant to his content. We could challenge these results but training on tweets by other republican politicians or supporters. Since the vocabulary would then be more homogeneous and the same content would be shared across positive and negative samples, we predict that the Naive Bayes' classifier would fail to fit. The Naive Bayes' assumption would break down as individual words would no longer characterize tweet authors. Then, we attempted to generate our own tweets mimicking the President's dialect with a LSTM-RNN and got an accuracy of 0.64. Our generator fooled the classifier well, but a Turing test revealed that humans would only accept 34% of our tweets. As our coherence metric lies at 54%, only 18% of our tweets are actually competitive. A principal component analysis of the embedding matrix showed interesting results in projecting terms relating to the election to the space in between words referring to a public office and words referring to news. The main benefit of the model architecture we used for tweets generation was its simplicity, which allowed us to iterate faster. However, its main drawback is that about half of the generated texts are still nonsensical or at least grammatically flawed. A really fun extension of this work would be attempting to train a Transformer model for this task.

Even though President Trump has his social media director Dan Scavino writing out some of his hot, medium, and mild tweets, the common grammatical and spelling errors his account is often mocked seem to be a crucial part of their appeal and success. His staff argues that "his unvarnished writing, poor punctuation and increasing profanity on Twitter signals authenticity." Well, at least we know it's distinguishable.

REFERENCES

[1] P. Bojanowski, A. Joulin, and T. Mikolov, "Alternative structures for character-level rnns," *CoRR*, vol. abs/1511.06303, 2015. [Online]. Available: http://arxiv.org/abs/1511.06303

[2] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," 2014.

[3] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013.

[4] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," *arXiv preprint arXiv:1802.05365*, 2018.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[6] C. Zacharias, "TWINT - Twitter Intelligence Tool," 2018.

[7] C. Hutto, "VADER - Valence Aware Dictionary and sEntiment Reasoner," 2016. [Online]. Available: https://github.com/cjhutto/vaderSentiment

[8] K. McKeown, *Text generation*. Cambridge University Press, 1992.

[9] D. M. Montesinos, "Modern methods for text generation," 2020.

[10] D. Pawade, A. Sakhapara, M. Jain, N. Jain, and K. Gada, "Story scrambler-automatic text generation using word level rnn-lstm," *International Journal of Information Technology and Computer Science (IJITCS)*, vol. 10, no. 6, pp. 44–53, 2018.