Quotation Construction: A Generative Approach

CS 6700: End of Semester Report

Chris Fifty, Jingqi Zhou
{cjf92, jz493}@cornell.edu

20 May 2018

"I every do not consul."

"I want $ in calls"

"Every man can tell how many goats or sheep he possesses, but to defend them negligently is nothing short of criminal."

Project Overview

The goal of this project is to use Artificial Intelligence and Machine Learning methods to model human language. More specifically, our team's goal is to build a system that generates quotations that an individual was likely to have said, based on a corpus of text from their written works or recorded speech. For example, our algorithm could generate a quotation for Harry Potter − one which is in line with his character, but was never actually written in the books. It is our hope that the sentences generated with our language model will be indistinguishable from those said by the actual individual we model.

Project Methods

N-gram Probabilistic Language Models:

Our baseline system will consist of a simple bigram and trigram language model. Bi/trigrams are generative models which construct sentences based on the word frequency within a corpus. For the bigram model, the conditional probability of saying word w_i given its precreding word w_{i-1} is:

$$P(w_i \mid w_{i-1}) = \frac{count(w_i, w_{i-1})}{count(w_{i-1})}$$

For the trigram model, the conditional probability of generating word w_i given its precreding words w_{i-1} and w_{i-2} is:

$$P(w_i \mid w_{i-1}, w_{i-2}) = \frac{count(w_i, w_{i-1}, w_{i-2})}{count(w_{i-1}, w_{i-2})}$$

While the inherent probabilistic definition is simple, several challenges arise when using n-gram language models:

Challenge 1: Handling Unseen N-grams

During evaluation on a held-out test corpus (approximately 10% the size of the training corpus), a n-gram combination may appear which was not present in the training corpus. For example, the bigram combination of *went running* may appear in the test corpus whereas the training corpus only contains the phrases *went swimming* and *went hiking*. This challenge is exacerbated when working with a small corpus.

The standard method of handling unseen n-grams is smoothing (i.e. moving probability mass from the most common bigrams to unseen ones). Accordingly, our bi/trigram language models implement Simplified Kneser-Ney smoothing to move a variable amount of probability mass dependent on the frequency with which a n-gram occurs. Kneser-Ney smoothing is widely regarded as the state-of-the-art smoothing method as its performance rivals hierarchical Bayesian Models methods and is relatively compact and easy to implement[1].

Challenge 2: Handling Unknown Words
A test corpus may contain words which are 'unknown' to the training corpus. In other words, the test corpus contains one or more words which are not in the vocabulary of the training corpus. A common technique to handle this difficulty is to create a placeholder word 'UNK' in the n-gram language model, and map all rare words (determined by word frequency) within the training corpus to this token. All occurrences of rare words in the training corpus are then replaced with the 'UNK' token, and training occurs normally. When a word which is not in the vocabulary of the model is encountered in the testing corpus, the unknown word is converted to an 'UNK', and evaluation continues. When an 'UNK' token is generated during quotation construction, one of the rare words is selected uniformly at random from the set of rare words in the training corpus.

Software Written:
All n-gram probabilistic language modeling software was written from the ground up, and no external libraries or APIs were used except to perform basic operations like reading in values from a csv file, parsing sentences into tokens, etc. The code for the n-gram language models can be found in the *bi/trigram* folder of the submitted code.

Recurrent Neural Net Language Models:
The second language model our team developed was Recurrent Neural Net based language model. Recurrent Neural Nets appear ideal for language modeling tasks, and in particular sentence generation. Rather than looking back one word, or even two words, a Recurrent Neural Net considers the entire preceding sentence when predicting the next word. For example, the sentence:

$$I\ grew\ up\ in\ France,\ so\ I\ am\ fluent\ in\ \_\_\_\_\_$$

is difficult for a n-gram language model to handle, since it only looks at the past n-words, and increasing n past 3 or 4 often leads to very poor performance due to overfitting and an inability to generalize. On the other hand, a RNN should be able to learn the dependencies between a location (i.e. *France*) and predicting a language (i.e. *French*) for this particular sentence.

[1] http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.161.3329

Numpy Implementation:
Following an online tutorial[2], our team wrote a Numpy implementation of a Recurrent Neural Net from the ground up, trained it on 186 Cicero quotations taken from BrainyQuote, and adapted the model to construct sentences based on the next word probabilities produced by applying a softmax function to the output layer.

The Numpy implementation uses a one-hot vector encoding for word vectorization, and as a result, the size of the input and output layers is equal to the size of our vocabulary. The model also consists of a single hidden layer of 100 dimensions. We chose a single hidden layer to reduce the training time of our model in addition to reasoning that we would need a larger volume training data to train a multi-layer RNN. While we had neither the time, nor the resources, to tune the model's hyperparameters, the tutorial suggested a learning rate of 0.005 with a single hidden layer of size 100, a tanh activation function, and random initialization of weights between the interval of $[\frac{-1}{\sqrt{n}}, \frac{1}{\sqrt{n}}]$ (n is the number of incoming connections from the past layer). These parameters are derived from Andrej Karpathy's[3] user-friendly and highly-cited blog about using RNNs to create character-level language models[4].

We determined the training time for each model by examining the loss after each epoch. We ceased training the model after the training loss remained steady for several epochs. We used a cross-entropy loss function as this loss function is commonly used and highly evaluated in language modeling academic literature:

$$L(y, o) = -\frac{1}{N} \sum_{n \in N} y_n log\ o_n$$

N is the number of training examples (words in our document), y is the correct label, and o is the prediction of our model.

The Numpy RNN produced very poor results, and we postulate that we would need more than 187 quotations to train a strong model. As a result, we increased the size of our Cicero training corpus to 4,000 sentences; however, this resulted in the numpy implementation training very slowly. We decided it was time to investigate optimized Deep Learning libraries which are adapted to run efficient matrix operations on a GPU.

Theano Implementation:
In addition to walking through the math and concepts behind basic RNN functions (forward propagation, stochastic gradient descent, backpropagation through time, etc.), the tutorial

[2] http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/
[3] Former PhD at Stanford specializing in Deep Learning and now director of AI at Tesla
[4] http://karpathy.github.io/2015/05/21/rnn-effectiveness/

referenced in the *Numpy Implementation* section also provided a Theano implementation of a RNN. To maintain consistency, we used the same hyperparameters for the Theano RNN that we used in the Numpy RNN. To more efficiently train our language model, we adapted the provided Theano RNN to fit our use case, set up a p2.xlarge server with a Nvidia K80 graphics card on AWS, and trained our models remotely. Throughout this project, we spent approximately $56 in AWS credit for remote training.

After training the Theano implementation for approximately 15 hours, we exported the saved parameters and analyzed the results. Once again, we found the generated sentences to be poorly formulated, and evaluation of a held out test set showed the perplexity[5] score of our model to be very high. This combination suggests that our RNN language model implementation fails to properly capture and internalize the dependencies between words in both the Cicero and Trump training corpuses.

Vanishing Gradient in Backpropagation Through Time:
The problem with RNN language models is the vanishing gradient in the backpropagation through time (BPTT) update. Since our activation function has a derivative of 0 at its tails, when a neuron is saturated, it will drive the gradients in previous layers to 0 during the BPTT algorithm. Moreover, the length of our unrolled neural net is variable across examples and equal to the number of words in a particular sentence. Accordingly, the backpropagation through time algorithm coupled with SGD struggled to properly update the model parameters for longer sentences, especially those within the Cicero training corpus. The vanishing gradient challenge results in the network's inability to learn long-term dependencies and reduces the overall efficacy of RNNs to create language models.

LSTM and GRU Neural Net Models:
Long-Short-Term Memory and Gated Recurrent Unit Neural Nets are specialized Recurrent Neural Nets which address the vanishing gradient issue by introducing a gate structure to manage the internal memory at each timestep of our RNN. The gating mechanism helps the Neural Net construct a new hidden state at each time step by deciding what information to keep (i.e. learn relevant long-term dependencies) and what information to discard (i.e. forgetting non-relevant information).

LSTM:
While our team did not implement a LSTM based language model, the LSTM paradigm is perhaps the most well-known language modelling neural net architecture and is relevant for motivating the creation of the Gated Recurrent Unit structure which we used in this project.

[5] an intrinsic method for evaluating the strength of a language model

LSTMs introduce a the concept of a ‘memory’ to the RNN paradigm and were first developed in 1997. Rather than the network being defined in terms of an input, hidden state, and output at time step *t*, a LSTM network consists of an input, output, hidden state, and memory for every time step. A LSTM has three gating mechanisms: the input gate, forget gate, and output gate. By considering the past hidden state and the current input, the input gate creates a candidate hidden state *g* at time step *t*. The forget gate determines how much memory from time step *t-1* is carried over into time step *t* (i.e. what to ‘forget’ from the last iteration memory), and as a result, the memory at time step *t* is a combination of the candidate state at *t* as well as the memory that passes through the forget gate. The hidden state at time step *t* is computed from a combination of *g* and the current time step's memory applied to the output gate.

The facet of LSTMs which make them so powerful is learning the parameters for each of these gates. Given a sufficiently large data set, a LSTM can model long term dependencies by learning the parameters for each of its gates in addition to normal RNN parameter updates with stochastic gradient descent.

GRU:
A GRU is a simplified, lightweight adaptation of a LSTM model first developed in 2014. It is proven to be just as powerful as a LSTM and consists of only two gates: a reset gate and an update gate. Similar to a RNN, a GRU also does not abstract an internal memory layer at each time step. Rather, the input and forget gate of an LSTM are combined to form the update gate of the GRU, and the output gate of the LSTM is disregarded entirely. The update gate helps decide how much information from the input and previous hidden state should be retained in the hidden state at the current time step. The reset gate is applied directly to the hidden state at time step *t-1* and determines how much of the past hidden state is forgotten. While a GRU architecture appears to be less complex than that of a LSTM, literature suggests that GRUs based language models are faster, more lightweight, and have roughly similar performance to LSTM based language models[6][7].

Software Written:
Our team adapted a Theano GRU based language model for both Cicero and Trump Corpuses[8]. While the downloaded model already contained the logic for essential neural net functions, we augmented the implementation by adding an embedding layer to plug in word embeddings as well as modified the code to calculate perplexity.

[6] https://arxiv.org/abs/1412.3555
[7] http://proceedings.mlr.press/v37/jozefowicz15.pdf
[8] http://www.wildml.com/2015/10/recurrent-neural-network-tutorial-part-4-implementing-a-grulstm-rnn-with-python-and-theano/

The Theano GRU model utilized rmsprop over stochastic gradient descent because the rmsprop algorithm assigns frequently occurring features a smaller learning rate and features which rarely occur a larger learning rate. This technique allows the model to variably adjust its learning rate to better preserve frequently occurring features and more quickly learn rarely occurring features. The other hyperparameters include a learning rate of 0.001 with decay, a hidden layer size of 128, and two GRU layers (i.e. two hidden layers). Adding a second GRU layer should allow the model to capture more complex interactions amongst the words in each sentence, and the intuition behind adding an embedding layer is to give our model a head start in learning how one word in our vocabulary relates to another. The word embeddings were learned for our corpus with word2vec's word embedding library.

Data Collection

Cicero Dataset:

The Cicero quotes were compiled from BrainyQuote, a site that aggregates quotations from famous individuals. Initially, we collected 186 quotations exclusively from BrainyQuote to train the bigram and trigram language models. However, this corpus size was too small for deep learning based language models. To supplement the Cicero corpus, we accessed Project Gutenberg's "Republic of Cicero," "Treatises on Friendship and Old Age," and "Letters of Marcus Tullius Cicero." For a final augmentation, the entire speech "For Marcus Tullius" (plaintext from Wikipedia) was also included in the corpus.

To increase the ease of word vectorization, improve data content, and decrease the vocabulary size, section numbers, headings, line numbers, and annotations made in the text were removed before addition to the accumulator csv. Moreover, in order to decrease the in-text variability with the quotations from BrainyQuote, British English was converted to American English. Finally, Python's Natural Language Toolkit (NLTK) was used to split the passages into sentences, and the quotations were stored in a csv file for ease of use.

The Cicero Corpus covers a large subject area − including but not limited to speeches, formal manuscripts, and daily letters −and consists of 4,354 unique sentences.

Donald Trump Dataset:

Similar to the initial methods employed to collect Cicero quotations, 577 Donald Trump Quotations were compiled from BrainyQuote. Similar to the challenges we experienced when dealing with a small dataset for the Cicero Corpus, the Donald Trump corpus also needed to be expanded. We decided to supplement the initial dataset with Donald Trump tweets which were accessed from www.trumptwitterarchive.com (a website which archives all Twitter posts made by Donald Trump).

As the dataset of Trump tweets consists of 33,676 total tweets as of May 19th, we chose to include tweets which contained one or more of the following keywords: {big, sad, good, bad, rich, smart}.

Upon comparison with the initial quotations from BrainyQuote, Donald Trump's tweets were found to be very different, in both formality and format nature. This disparity is caused by Twitter's informal atmosphere, the inclusion of hashtags, and the '@user' retweet word. In order to develop a language model on a corpus markedly different from the Cicero corpus, we decided the Twitter dataset would replace the Trump BrainyQuote quotations entirely as our Trump Corpus. The final Twitter dataset for Donald Trump consists of 2,379 sentences.

Additional Notes:
Both datasets used non-ascii characters which is problematic for word vectorization. The non-ascii characters were identified with the use of regex and were either removed outright (i.e. an emoji in a tweet) or converted to an ascii equivalent (i.e. æ to ae).

Evaluation

Perplexity:

Perplexity is an intrinsic evaluation technique and one of the most commonly used metrics to determine the strength of a language model. It is defined as:

$$PP = \left(\prod_i \frac{1}{P(W_i|W_{i-1},\ldots,W_{i-n+1})}\right)^{\frac{1}{N}}$$

Where N is the number of tokens in the test corpus. The perplexity is computed on the test set, and provides a quantitative measurement of how well the language model can predict the next word in a never-before-seen sentence. Therefore, perplexity not only conveys a sense of the strength of the language model in capturing corpus-specific patterns, but also the generalization abilities of the language model on the test dataset. Perplexity can also be interpreted as a function of the average (per-word) log probability. Taking the log of the above equation:

$$PP = e^{\frac{1}{N}\sum_i -log\, P(W_i|W_{i-1},\ldots,W_{i-n+1})}$$

Accordingly, perplexity can be used to evaluate probabilistic language models against those developed by deep learning if both models are trained on the same training data and evaluated on the same test corpus.

Results:

<u>Cicero Corpus:</u>

The Cicero corpus consisted of approximately 4,400 complex sentences, and the dataset was split at random into 4,000 sentences for the training corpus and 400 sentences for the test corpus. The following sentences are several examples from the training corpus:

1. "The very philosophers themselves, even in those books which they write in contempt of glory, inscribe their names."
2. "For our country has not produced us, or educated us under a law, that she is entitled to no support on our part, lending herself as it were to our convenience only; furnishing a secure refuge, and a tranquil and peaceful asylum to our indolence: but rather holds as pledges to her, to be employed for her benefit, the many and great faculties of our mind, genius, and reason; and only permits us to appropriate to our private purposes, that portion of them, of which she stands in no need."
3. "I confess that they, if they be not deliverers of the Roman people and saviours of the republic, are worse than assassins, worse than homicides, worse even than parricides: since it is a more atrocious thing to murder the father of one's country, than one's own father."

The perplexities are as follows:

Language Model	**Perplexity**
Bigram	358.65
Trigram	139.64
Recurrent Neural Net (Theano)	773.08
GRU Neural Net (Theano)	116.85

<u>Cicero Perplexity Analysis</u>:

From reading academic papers analyzing and comparing a variety of language models for the past semester, these results are exactly what we would expect to see. A trigram is expected capture the semantic structure of a language to a higher degree than a bigram without suffering from the overfitting nature of higher-order n-grams.

We see that the RNN is unable to capture even simple inter-word dependencies. This result can be explained by having a simplified RNN based language model (only one hidden layer), and not tuning the hyperparameters of the model. Nevertheless, literature suggests that RNN based language models perform poorly, especially on complex datasets like the Cicero Corpus.

Finally, we see that the Gated Recurrent Unit based language model surpasses the competition and is the state-of-the-art method which recent literature purports it to be. The added complexity of the gating mechanisms − in addition to the inherent RNN structure which allows learning of arbitrarily long inter-sentence word dependencies − causes this method to perform the best on intrinsic evaluation and crowns the GRU as the king of modern language models.

Trump Corpus:
The Trump corpus consists of approximately 2,600 sentences from Donald Trump's tweets, and the dataset was split uniformly at random into 2,350 sentences for the training corpus and 250 sentences for the test corpus. The following are several examples from the corpus:

1. "I don't want to be the only billionaire in America---I want all Americans to be rich."
2. "WHAT WILL WE GET FOR OUR LIVES AND $ BILLIONS?ZERO Happy Father's Day to all even the haters and losers!"
3. "Sad!"

The perplexities are as follows:

Language Model	**Perplexity**
Bigram	236.32
Trigram	431.52
Recurrent Neural Net (Theano)	463.66
GRU Neural Net (Theano)	315.38

Trump Perplexity Analysis:
It is very unusual for a trigram language model to have higher perplexity than its bigram counterpart. Upon further investigation, this result occurs because Donald Trump naturally tweets in bigrams. For example, Trump uses word combinations such as 'third rate', 'running for', 'republican party', and 'smarter than' in almost all of the tweets in our dataset. As a result, the bigram language model is particularly suited and incredibly successful at generalizing and creating Trump tweets since it naturally creates these bigrams. Moreover, the Trump dataset contains sentences which are poorly formed and the natural sentence structure is distorted with hashtags and retweet twitter handles. We postulate the inherent noise of this dataset could contribute to this result.

Aside from this anomaly, we see a similar pattern to the Cicero Perplexity table. The RNN based language model is unable to capture the inter-sentence word dependencies. The trigram does well, and the GRU language model performs better than the trigram. We postulate that even on the Trump Tweet dataset, a GRU with optimized hyperparameters could outperform the bigram language model since the two perplexities are relatively close and GRU RNNs are generally very sensitive to hyperparameter tuning.

Software Written:
Neither the Theano RNN nor the GRU language models computed perplexity. Accordingly, we had to extract the relevant probabilities from the output layer, apply a softmax function to normalize the output, and then code the perplexity calculation within the overarching Theano framework for both the Cicero and Trump models.

Extrinsic Evaluation:

<u>Cicero Corpus</u>:
Restricting our training data to the initial BrainyQuote dataset, we generate the following sentences using a trigram language model:

1. The rule of friendship is like insatiable longing to see the truth.
2. All pain is either severe or slight, it is foolish to tear one's hair in grief, as it is with our own luxury, our own luxury, our own criminality that we have to contend.
3. Every man can tell how many goats or sheep he possesses, but to defend them negligently is nothing short of criminal.

The generated sentences were less realistic when trained on the corpus of 4,000 Cicero sentences. Trained on the augmented dataset, our trigram produced the following sentences:

1. Do you ever hear of any public work ornament.
2. If that was no law at all, publius scipio, and injunctions, I ought to refer the matter placed up for sale, the hated hastening to his necessity.
3. What designs are more men ennobled by principles of my exceeding devotion to the tribes into decuries , that none could have done?

As we would expect based on the intrinsic evaluation, the RNN model produces poorly constructed sentences after training on the augmented Cicero dataset:

1. Messengers from men is appreciated seeing.
2. Describe the roman difficult fire that receiver demanding.
3. Lifetime my republic he beware the surprised method.

While the intrinsic evaluation metric indicates that the GRU based language model would produce the most ‘human’ quotations, this is not the case. Rather, it learns the correct inter-sentence pattern such as object-verb, adjective-object, and why-question-question mark, but lacks the ability to combine them in an intelligent pattern:

1. For themselves I will say against to the are beginning of a will I either discharge this in the concerning up in virtue.
2. What honour roman be senate to which titles ?
3. I every do not consul.

Trump Corpus:

Restricting our training data to quotations from Donald Trump's speeches (approximately 575 sentences), we generate the following sentences using a trigram language model:

1. Well, I'm competitive, and we have a great course.
2. Americans want great schools for their families, and they want to take jobs away from other countries.
3. I don't frankly have time either.

Considering an augmented training corpus consisting solely of Donald Trump's tweets (approximately 2,400 sentences), our trigram model generates the following sentences:

1. AOL has been a disaster!
2. Putin has become a sad day for America!
3. You mean the fact that he beat me on the scale of 0 to 10!

As Trump tweets are somewhat intrinsically chaotic, our RNN model has the capacity to generate tweets which may be mistaken for actual tweets:

1. twitter good!
2. laughingstock president food.
3. shots very fired continue liar and party disaster @ turbines change.

Similar to the RNN model, the GRU based language model can generate tweets that can be mistaken as being "true" Trump tweets; however, this method is better able to construct more complex and nuanced sentences which logically make sense:

1. I want $ in calls .
2. president obama recognition - sad!
3. sadly they are not smart?

Survey on n-gram Language Models:

Our team evaluated the quotations generated from the bi/trigram language models trained exclusively on the Trump and Cicero BrainyQuote corpus by asking friends and classmates to fill out two separate google quiz forms: one for Cicero quotations[9] and the other for Trump quotations[10]. The google quiz asked participants to differentiate between quotations that were said by the historical figure and quotations that were computer generated.

Each poll consists of a bigram and trigram section for a total of 24 total questions. Each question consists of a quotation and a 'computer generated' or 'Trump' (or alternatively 'Cicero' for the Cicero quiz) multiple choice response. Participants are asked to mark 'computer generated' if they believe the quotation is generated from a computer or to mark 'Trump' if the quotation is actually said by Donald Trump. The quiz provides the user with several sample quotations from Trump/Cicero to give each participant a sense for Trump/Cicero's word order typology. The first 12 questions relate to the bigram language model and the second 12 relate to the trigram language model.

Survey Analysis: Cicero

Regarding the bigram section of the Cicero quiz, the average is 66.3% correct. For the trigram part of the quiz, the average is 53.9% correct. Accordingly, the bigram results indicate that humans have a marginally higher than at-chance probability of determining if a quotation was computer generated or not. On the other hand, the trigram results indicate that humans have a nearly at-chance probability of determining if a quotation was generated with a trigram language model or said by Cicero. Twenty-one participants filled out this survey. The average score for the trigram language model was significantly lower than that of the bigram language model ($p = 0.0001186$, Paired Student's t-test). This statistical analysis quantitatively indicates the respondents had a harder time differentiating between trigram generated quotations and actual Cicero quotations.

Survey Analysis: Donald Trump

Regarding the bigram section of the Trump quiz, the average is 62.8% correct. For the trigram portion of the quiz, the average is 60.7% correct. These results indicate that participants have a slightly higher than at-chance probability of predicting if a quotation was generated by a computer (for both bigram and trigram language models) or said by Donald Trump. Twenty-four participants filled out this survey. There was no significant difference in terms of average score between the bigram and trigram language models.

Closing Remarks:

The goal of this project was to train a model to generate quotations which are indistinguishable from those in the training data. We see that the complexity and sentence diversity within the

[9] Cicero Google Quiz Link - https://goo.gl/WpF96R
[10] Trump Google Quiz Link - https://goo.gl/zzRic3

Cicero Corpus causes traditional and even state-of-the-art deep learning methods difficulty in mimicking Cicero's word order typology. However, an unexpected result occurred. Limiting the size of the corpus to a high quality dataset and purposefully overfitting with a probabilistic language model can yield spectacular results. Overfitting to a small corpus of "strong" data causes quotation generation to amalgamate partial sentences in the training data. For example, the trigram quotation:

Every man can tell how many goats or sheep he possesses, but to defend them negligently is nothing short of criminal.

Is constructed from these two sentences in our training data:

Every man can tell how many goats or sheep he possesses, but not how many friends.

It might be pardonable to refuse to defend some men, but to defend them negligently is nothing short of criminal.

Rather than learning the underling inter-sentence word dependencies of Cicero, the trigram language model can seamlessly mix and match different sentences in our training data to produce high-quality human-like quotations. As this method is incredibly lightweight, fast, and easy to implement, future research into language modeling may want to consider further investigating overfitting to small datasets in generating sentences which are indistinguishable − yet different − from those in the the training data.

Switching focus to the Trump dataset, the inherent chaotic nature of tweets lends our language models leniency when generating sentences. In this domain, we were able to develop a language model which produces tweets which are indistinguishable from those written by Donald Trump. Moreover, the intrinsic and extrinsic evaluation metrics are closely linked: GRU language model generated tweets outperform the trigram generated tweets, which in turn outperforms the RNN language model generated tweets. The GRU language model consistently creates complex, and logically constructed tweets at a higher rate than either the trigram or RNN language models. Lastly, the GRU language model was able to produce some rather comical sentences which make sense logically (but would never be tweeted by Trump), and others which are right in line with his twitter history.