Using LSTMs for Text Generation with Style Transfer

Hardik Paliwal, Da Peng, Nikhil Shende
{hpaliwal, d7peng, nnshende}@uwaterloo.ca
University of Waterloo
Waterloo, ON, Canada

Introduction

We are creating a text generation network, with style transfer focused on Twitter users' tweets. The end goal is to be able to generate text imitating tweets from any user on Twitter, given a prompt.

The problem is interesting as text generation neural nets are being applied at an ever increasing rate in the real world. One example would be the rising presence of chat bots on many companies landing pages. Style transfer (which is essentially what we are doing) could be highly useful in, for example meeting corporate standards on how to communicate with customers. Another example could be noticing overused words and phrases by a politician which the model should pick up.

We picked Twitter as the medium and not other datasets (ie, transcripts of speeches, Reddit, newspapers etc) as it is highly used by the general public not only certain important individuals, has well written, popular api's for data gathering, and its own unique internet subculture which should be reflected in our model's grammar.

The research question our paper will address is the following: out of the LSTM and Transformer neural network models, which is better at generating text that, from a human's perspective, matches the style of the text it was trained on? At a high level, LSTMs (Long Short-Term Memory) are a type of RNN that keeps track of long-term dependencies and state in the network, and transformers are feed-forward networks made of encoder/decoder pairs that can help "pay attention" to different parts of the input. We will create our own LSTM network, and use GPT-2 for our transformer model. Then we will train them both on the same data from Twitter, and use both human evaluations and automatic evaluation methods to compare the two on syntactic and semantic correctness, as well as style transfer. Specifically, for human evaluators, we will ask fellow university students to fill out an online questionnaire to evaluate the coherence of the generated tweets, as well as compare the LSTM and transformer-generated tweets to choose which one "fits in" better with the real tweets to judge style transfer. We also complement these human evaluation methods by using an F-score metric to compare the style transfer of the generated tweets from the two models.

Related Work

Recurrent neural networds (RNNs) have proven to be effective in the field of text generation. Specifically, long short-term memory neural networks (LSTMs) are a type of RNN that have the advantage of not suffering from the vanishing gradient problem, and are especially effective at text generation. Wen et al. have found that using LSTMs to generate text at the word-level can produce results which human readers score as "higher on informativeness and naturalness" compared to using other systems (Wen et al. 2015). Additionally, Sutskever et al. have found that RNNs are also effective at generating text at the character-level (Sutskever, Martens, and Hinton 2011). Tikhonov and Yamshchikov used a version of LSTMs "with extended phonetic and semantic embeddings" (Tikhonov and Yamshchikov 2018) and tested it under the Bilingual Evaluation Understudy (BLEU) system, and found that the text generated by their system was highly accurate in relation to human text.

Ideally, a Twitter bot should pick up on writing styles of different Twitter users. Potash et al. have found that LSTMs can effectively imitate speaker styles in the context of rap lyrics (Potash, Romanov, and Rumshisky 2015). The lyrics generated with their LSTM model performed better on style matching metrics than a baseline model. Han et al. have made similar findings about style transfer in text generated using an LSTM model. With their model, they found that "style transfer can be conducted not only at the lexical, phrasal or syntactic levels, but also on a higher level, such as author's individualized writing style and genre preference." (Han, Wu, and Niu 2017). This finding is highly relevant to a tweet generation bot, because a tweet generation bot should ideally have style transfer on these higher levels, so that they can match Twitter user in terms of writing styles and topics discussed.

In recent years, transformers have emerged as a language generation tool that could be more effective than LSTMs. Transformer models have the advantage that generation of each token (word or character) does not depend on the last. Therefore, text generation and training of the neural network can be done in parallel, which is much faster and takes

advantage of modern GPUs' ability to process in parallel. Danyang and Gongshen Liu have found that using a combination of the transformer model and an LSTM network produces results that have better semantic coherence than other models (Liu and Liu 2019). However, in order to meet our time constraints of this project, we have decided to stick with LSTMs, which we feel we can find more information on on the internet.

Lastly, we have to discuss GPT-2 (Radford et al. 2019), the landmark transformer language model published in 2019. While the multi-task nature of GPT-2 is far beyond this project, an alternative approach to this project can be retraining GPT-2 on Twitter data in order to accomplish the style transfer while GPT-2 itself provides the majority of the language model. This approach has been successfully used in numerous projects, for example: ai.dungeon.

Methodology

With our project we will be looking at two different neural network models: Transformers (GPT-2) and LSTMs.

Our rationale for choosing LSTM as one of the models to compare is that it has shown significant promise in past studies not only for generating coherent text in general that is high on "informativeness and naturalness" (Wen et al. 2015), but also for transferring styles (for example, writing styles of authors (Han, Wu, and Niu 2017), and rappers (Potash, Romanov, and Rumshisky 2015)). Therefore, we feel that LSTMs should be effective at generating Tweets that are informative, natural, and that imitate a specific Twitter user's tweeting style.

Our rationale for choosing GPT-2 as our second machine learning model is that it has already been trained on massive amounts of data, and has consistently been shown to generate "state of the art results" when used for language generation (Radford et al. 2019). Additionally, like the LSTM model, GPT-2 has also shown significant promise in imitating styles when generating text. As Wang et al. put it, "The GPT-2 model, pretrained with massive unlabeled corpora, is able to capture the generic knowledge of language and can be adapted to formality style transfer." (Wang et al. 2019)

Now we will explain both algorithms at a high level. First we explain LSTMs, or "Long Short-Term Memory" neural networks. LSTMs are a variant of RNNs that can keep track of long term dependencies/state of the network in such a way to address the "vanishing gradient problem".

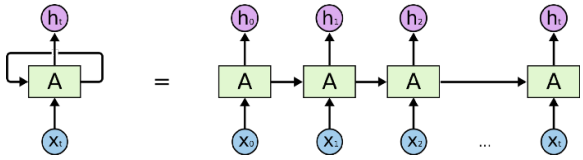

Figure 1: Diagram of an RNN

RNNs, or recurrent neural networks, are similar to feed-forward neural networks in the sense that they consist of a series of layers of nodes that generate an output vector from an input vector, but unlike feed-forward networks, they also re-use the output as input for the next output generation step. The equation for RNNs is : $h_t = f(h_{t-1}, x_t)$ where h_t is the output for the t^{th} node in the network, f is some arbitrary activation function, and x_t is the input for the t^{th} node.

The vanishing gradient problem occurs because when training an RNN using back propagation, the gradients which are back propagated can quickly become very small after just a few steps in the network. This leads to the the model "forgetting" long term dependencies, and is biased towards its most recent inputs. LSTM's attempt to limit this problem by having multiple layers or cells to learn when to pass information on.

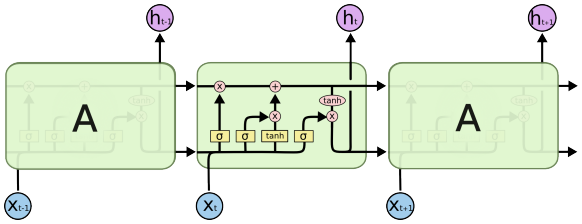

Figure 2: Diagram of a LSTM chain

In LSTM, there is a "chain" containing a "cell state" which runs through the entire chain of modules and is potentially modified in each module. This chain serves to persist meaningful state across the modules as long-term memory. Each module is able to remove or add information to the cell state by the use of "gates". There are 4 gates in each cell, denoted mathematically as $\{f, i, \tilde{C}, o\}$, and often called the "forget gate layer", the "input gate layer", the "tanh layer", and the "filter layer", respectively. At a very high level, the purpose of each gate is as follows: the forget gate layer chooses what information to transfer from the previous module's cell state to the current cell state; the input layer gate and tanh gate work together to determine which information from the previous output should be transferred to the cell state; and the filter layer determines which part of the current cell state should be reflected in the output for the current module. Now we discuss the mathematical details of each gate. The first layer in an LSTM module, f, is a sigmoid layer. This step can be expressed with the equation: $f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$. The result of this equation is connected to a gate which multiplies the cell state. So it will either multiply the cell state by values ranging from 0 to 1. The next two layers, i and $\tilde{C}$, are a sigmoid layer and a tanh layer. Their equations are as follows: $i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$ and $\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$. The results of these two equations are then multiplied by each other and then added to the cell state. This value of the cell state will be output from the module; the equation is: $C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$. The last layer, of a module in LSTM is a sigmoid layer. The equation is: $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$. Now what is actually output is the variable h_t, the equation for which is: $h_t = o_t * \tanh(C_t)$. That essentially concludes the description of what is going on within one of the modules in

the network. Then the h_t values will result in the output; in the given scenario of text generation the output will be a sequence of characters. The length of the chain of modules will be given as a parameter.

The other architecture that we will be exploring is the transformer model. Transformers, as a whole, are a result of the realization that most state of the art NLP algorithms were gated RNN's (such as LSTM) with attention mechanisms, and that these attention mechanisms are powerful enough, without the sequential processing, to achieve the results of RNN's with attentions. Keeping the "attention" part while removing the "recurrent" part of these existing networks allowed transformers to be trained in parallel, which is naturally much faster than training sequentially.

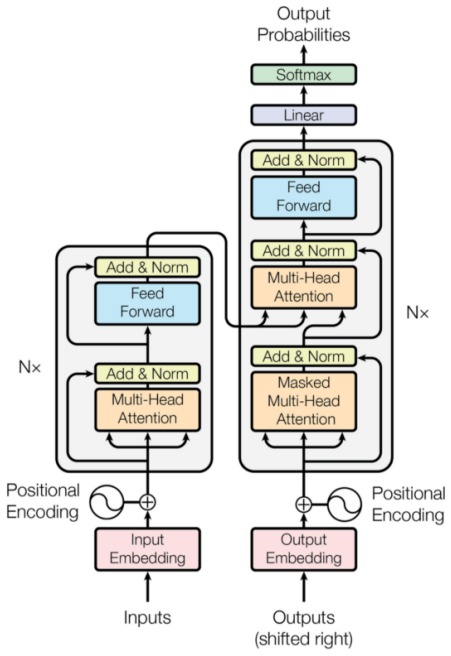

Figure 3: Transformer Architecture

Transformers consist of modules of encoders and decoders, both of which can be stacked on top of each other multiple times. Both encoders and decoders consist of a feed-forword networks for processing inputs and attention mechanism layers, which for every input, weighs the relevance of every other input to determine the output. Lastly, because Transformers have no conception of state (unlike RNNs), the position of every word/part in the input is added to the vector embedding of each word/part. (Vaswani et al. 2017)

For this project, we will specifically look at the GPT-2 transformer model by OpenAI. It is a language model, like the predictive text you would see on smartphones keyboards, just one that is far larger and more advanced. As described on the OpenAI website:

> *"GPT-2 generates synthetic text samples in response to the model being primed with an arbitrary input. The model is chameleon-like—it adapts to the style and content of the conditioning text. This allows the user to generate realistic and coherent continuations about a topic of their choosing, as seen by the following select samples."*

However like another popular transformer model BERT (Devlin et al. 2018) which only uses encoders, GPT-2 only uses decoders. In a general sense, how GPT-2 works is that the previous token will be sent through all the decoders to generate a list of most likely successors. The successor with the highest probability is chosen.

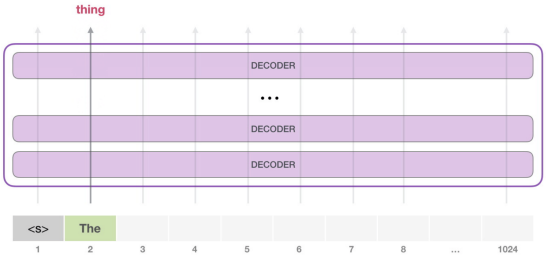

Figure 4: Diagram of GPT-2 decoders generating the next token

There are two hyperparameters in GPT-2 that are worth mentioning. The first one, top-k, addresses a problem where the algorithm can get stuck generating an infinite loop where a sequence of particular tokens will simply generate each other and the algorithm will keep cycling through them. Top-k determines the number of previous tokens to look at to generate the next token. For example, if top-k = 5, then the algorithm will take the previous 5 tokens into account when generating the next token, rather than the last one. When top-k is large, it is extremely unlikely that the same tokens will be constantly generated by the last k tokens. The second hyperparameter worth mentioning is called the temperature. The temperature determines the "randomness" in the resulting output. A lower temperature means that the output will be more coherent and on-topic, but may repeat itself.

Note that there are different versions of GPT-2, sized depending on how many decoders there are. For this project, we will be using the 335M parameter version as (Case 2020) indicates that it is not significantly worse than the 1.5B parameter version whilst being far easier to train (the smaller version still takes up 1.5G of ram!).

Datasets

We will collect the data used to train our models through Twitter's public api, with the help of the Tweepy python library. We decided to collect the data ourselves rather than using existing datasets because Tweepy makes this process relatively easy, and we will get the most recent tweets compared to third party data.

Our first dataset will consist of tweets from Donald Trump. He is picked because of his high profile, unique style of writing, and a mind boggling large number of tweets

(54270 at the time of this writing) which means a decently sized data set to train our models.

As most twitter users do not meet the previous criteria, the second dataset will instead be a category of users. We picked news organizations as the category due to similar style, content and large number of tweets. This category will contain tweets from @cbcnews, @cnn, @nytimes, @theeconomist, @bbcworld, @ap and @reuters, combined for total of 2.3M tweets.

Note that we are collecting only English language data as training GPT-2 on different languages (with a different character domain than English) will require us to retrain it from scratch (which is far beyond the scope of this project).

Lastly, we have to discuss prepossessing. Once the tweets are collected, the contents of each tweet will be stored 1 tweet per line in a text file. All metadata such as timestamp and author will be dropped. Then this categorical, string data has to be converted into integer vectors for the network to work with.

First, we have to tokenize the dataset. For LSTM we used a word-level mapping rather than a character-level mapping because with some preliminary tests, we found by inspection that the word-level LSTM performed significantly better than the character-level one. In order to account for the language used in Twitter, which often includes hashtags and slang, we gathered the words that were most commonly used in the training set and used those as our domain. Specifically, we used any word that appeared more than 5 times in the training set. We then 1-hot-encoded every tokenized word, giving a vector for each word. As For GPT-2, since the model already uses byte-pair encoding and has a character domain of 50000+ english language sub-words, the decision to use byte-pair encoding is already picked for us.

Results

Experimental Design

Our end goal is to see which of the two machine learning models—LSTM and transformer models—are better at generating coherent text with style transfer. In order to answer this research question, we implement both of these algorithms, train them on the same dataset, and measure their effectiveness on syntactic and semantic correctness, as well as on style transfer from the original tweets.

As described in the Methodology section, we will implement our own LSTM neural network, while using the pre-trained GPT-2 library as our transformer model. All code will be run in Google Collab. In terms of libraries, we will be using Keras (for our LSTM implementation), Numpy, and the GPT source code located at https://github.com/openai/gpt-2.

For each dataset (as described in the Methodology section), we will split our available data into training, validation, and test data with the standard 80/10/10 split. We choose this because it is a standard and reliable choice. We decided that k-fold cross-validation would not be necessary, because we feel our datasets are sufficiently large (54K tweets and 2.4M tweets) so that having a 10% validation set is sufficiently large, and will take much less training and validation time than a k-fold cross-validation.

We will use a batch size of 32. This is a safe, commonly used choice, and we will test different batch sizes once we implement the models to see what is optimal for our models. We also feel that 32 is a good balance between training speed and convergence speed. Next for epoch size, we will choose 10. We chose a relatively small number to begin with because our large datasets means that a large number of epochs should not be necessary (and would be expensive).

Lastly for GPT-2, top-k will be set to 40 (default value) and temperature will be set to 0.7 (also the default value). Both are values commonly used with strong results by other studies using GPT-2.(Case 2020)

After choosing these initial hyperparameter values, we will fine-tune them with our models to minimize cross-entropy. As will be discussed shortly, cross-entropy is not our final measure for performance of the models. Instead we are using a combination of human evaluations and automatic evaluation methods. However, it would take way too long to evaluate our models with these final evaluation methods at every step to fine-tune our hyperparameters, so we must rely on cross-entropy to do this. Note that as we fine-tune the hyperparameters, they may also differ between the two models. For example, running through more epochs may be more beneficial for the GPT-2 model compared to the LSTM model.

Evaluation Methods

Once we have trained both models and optimized their hyperparameters to minimize cross-entropy, we can compare the two. In order to compare the LSTM and GPT-2 models, we measure them both on their proficiency in two different areas: syntactic and semantic correctness, and style transfer from the original Twitter user. We will use our test set to evaluate our model.

First, we measure their proficiency at generating syntactically and semantically correct text. To do this, we use a human evaluation method. We use human evaluation because, as Celikyilmaz et al. note, "the ultimate goal in NLG is to generate text that is valuable to people. For this reason, human evaluations are typically viewed as the most important form of evaluation for NLG systems and are held as the gold standard when developing new automatic metrics." (Celikyilmaz, Clark, and Gao 2020). To measure syntactic and semantic correctness, we present people with generated tweets and give them the following prompt:

> *Please rate, on a scale from 1 to 10, how understandable the following text is, with 1 being "I have no clue what this is message is conveying" and 10 being "I completely understand what this message is conveying"*

The higher the ratings, the higher we consider the correctness to be. Note that it is difficult to use automatic methods to evaluate syntactic and semantic correctness on tweets, because they often have slang or abbreviations, or do not follow traditional grammar rules. Therefore, a human evaluation is our best bet.

Second, we measure the models' proficiency in imitating the target author's tweeting style. In order to measure style transfer, we use a combination of the chrF algorithm and human evaluation. Again, we use a human evaluation because ultimately, the goal of our models are to generate text that is meaningful to humans, and with styles that humans consider to be close to the original Twitter user's tweets. However, this time it is possible to use automatic evaluations methods, which complements the human evaluation method because humans are prone to biases.

In terms of automatic evaluation methods of style transfer, there are a number of choices, with the most popular being the Bilingual Evaluation Understudy algorithm (BLEU), that was originally designed to evaluate translations by comparing n-grams in the generated translation and a human translation (n-grams are simply a sequence of n contiguous words or characters). However, upon learning that "recent work argues that although [BLEU] can be a good metric for the machine translation task ... for which it is designed, it doesn't correlate well with human judgments for other generation tasks outside of machine translation" (Celikyilmaz, Clark, and Gao 2020), we decided to use the chrF method. ChrF stands for character n-gram F-score algorithm. An F-score is a measurement of similarity between two texts, with the formula:

$$F = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Where precision is the fraction of n-grams present in the generated text that are also in the human-translated text, and recall is fraction of n-grams in the human-translated text that are also present in the generated text. The "chr" part of chrF simply means that we are comparing at the character (rather than word) level. The reason for comparing at the character level is because "it helps to better match the morphological variations in words" (Celikyilmaz, Clark, and Gao 2020). For example, "run" and "running" are matched for the first few characters but not as a whole word. For our purposes, we will use an actual tweet from the Twitter user as the "human translated" text, since we are not dealing with translation. The reason why we are choosing chrF is because it is has higher correlation with human judgements than other n-gram-based evaluation methods (McCarthy and Jarvis 2010).

For our human evaluations of style transfer, we will first present people with 10 tweets from the target Twitter user, with the following message:

> *Please read the following tweets from a Twitter user to get an idea of their tweeting style.*

Then, we will present them with a tweet generated either by the LSTM or GPT-2 model, and ask them:

> *Please rate, on a scale from 1 to 10, how well the following tweet matches up, stylistically, with the previous tweets you just saw. Please try to ignore any grammar mistakes, and only evaluate the style similarity.*

Overall, we expect that the GPT-2 model will be better at generating tweets that are more syntactically and semantically correct, because it has been trained on a 40GB dataset that we don't have the resources to train our LSTM model on. However, the LSTM model may generate text that is closer to the original tweets, because it is only trained on the data from the original tweet. Therefore, it has potential to have a better style transfer than the GPT-2 model. Additionally, due to the fact that LSTM is recurrent, it may be better at maintaining a consistent context within the tweet itself, meaning that each tweet is more likely to have a consistent theme.

Note to marker: This is where the section written for D3 begins

Design and Implementation Details

We used the same data sets to train both the LSTM and GPT-2 models. For Trump, we found a large, free dataset on Kaggle to use. For the news organizations Tweets, we used the Tweepy library to retrieve the data ourselves. We didn't use tweepy for Trump tweets because it is limited to the past 3000 tweets of a user, which is too small of a dataset to train on. We noticed that some of the data needed some preprocessing and filtering. For example, the Trump dataset included retweets, which we removed because they don't display Trump's tweeting style, which is what we are trying to imitate with our AI. Also, some of the tweets in the datasets were truncated with ellipses, so we removed those as well.

For the LSTM model, we used the Keras library to implement a word-level LSTM neural network. Initially, we wanted to use a character-level LSTM network, which generates one character at a time. However, after implementing this, we found that it had fairly poor results. Depending on the choice of hyperparameters, it would either generate gibberish (that didn't contain any real English words), or it would be extremely repetitive (as an example, one of the Tweets it generated was "Tomorrow at 7P.M, the president will a the president the the president the president the president the president ..."). We initially thought this was a result of a non-optimal configuration of hyperparameters, but after looking online for other peoples' results from using character-based LSTMs, we found that they had similar difficulties. Therefore, we decided to try a word-based LSTM and we ended up getting better results.

The input for a word-level LSTM is a sequence one-hot vectors, where each index represents a single word in the input. Therefore, in order to specify the vector size and choose which words corresponded to which indices, we had to choose a dictionary. The difficulty with using Twitter input data is that tweets often contain hashtags, abbreviations, or slang, which are not formal English words. Therefore, rather than using a list of commonly used words in the English dictionary as the input vector shape, we analyzed the training set to see which words appeared often. We first converted all words to lowercase (so that "Foo" would be considered the same word as "foo"), and then counted the frequency of each word. If the word appeared more than 5 times (we chose 5 as the threshold since it yielded a reasonably sized dictionary while ignoring rare words), we included it in our dictionary. If more than 80% of the words in any

tweet were not in our dictionary, we would remove that tweet from the training set. Otherwise we stripped the tweet of any words that were not in the dictionary. We chose 80% as the threshold because if it was any lower, the tweets with the stripped words would make little sense, and if it was higher than the remaining training set would be too small. We also noticed that many Tweets had URLs at the end, which were very rarely repeated between tweets, so we ignored those, since they would likely not be in our dictionary anyways.

For the model itself, the input was a 5xN binary matrix, where N is the number of words in our dictionary. The number 5 represents the number of words that we used to generate the next word. For example, our model might take in the 5 words "Listen to an interview with" and output "Donald". We figured that since it is an LSTM model, which should have some memory of past generated words, this number would not have to be very large, and indeed a few tests with larger values (10, 15) showed no decrease in cross-entropy.

The output of the model was a vector of floats of size N with larger values where the model predicted the word corresponding to that word more highly. We then used a temperature parameter to decide which word was chosen as the next word. If the temperature was higher, our model was likelier to pick a word with a smaller value in the output vector. We tested different values of the temperature and found that a temperature of 0.7 had the best balance between chrF and wordF scores (Figure 5), so we used it as our final temperature. Also, sometimes temperatures that were too low generated repetitive text (ex. the same two words over and over again), so we avoided low temperatures.

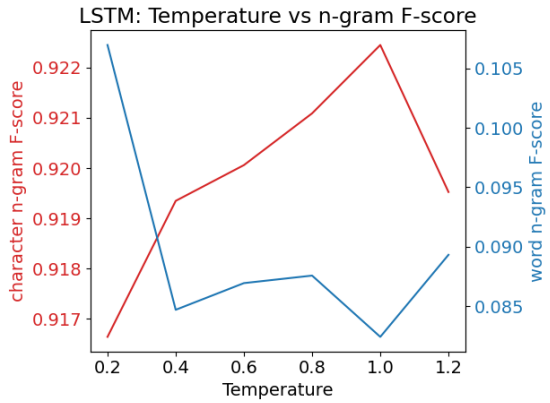

Figure 5: F-scores for the LSTM model. The F-scores using characters as n-grams is the graph in red, with the left y-axis scale. The F-scores using words as n-grams is the blue graph, with the right y-axis scale. When using word n-grams, F-score decreased as temperature increased, which is expected, because we are choosing words that are predicted as less likely by the model. We were not sure why, but when using character n-grams, the F-score seemed to increase as temperature increased: perhaps it is just due to a small sample size: after all, the differences were minimal.

For our LSTM model shape, we ended up using a model with 3 hidden layers, the first being an LSTM layer with 128 nodes, the second being a batch normalization layer, and the third being a dense layer with 128 nodes. Unfortunately, since our model took approximately 2 hours to train, we did not have a lot of time to play around with different layering methods. Instead, we looked at articles where other people made similar models, and tested out their parameters. Our general finding, from testing a few different layering methods, was that if there were too few layers or too few nodes on a certain layer, it would underfit, but if there were too many layers or too many nodes per layer, it would overfit and not generalize well (and also would take very long to train). Therefore, we tried to strike a balance between the two, and ended up with the model just described.

We also experimented with different numbers of epochs. Generally, as the number of epochs increased, the cross-entropy decreased and accuracy increased, but this improvement slowed as the number of epochs got closer to 10 (Figure 6), so we decided to use 10 epochs.

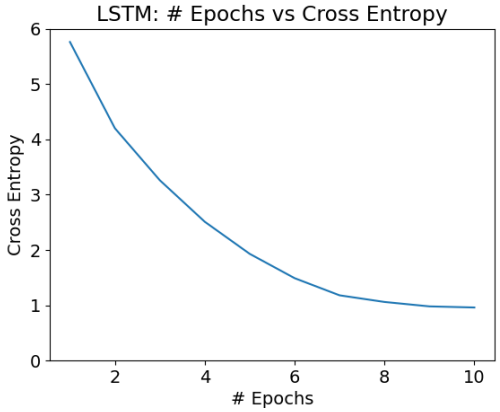

Figure 6: Cross-entropy of the LSTM model with different number of epochs. The cross entropy decreased as the number of epochs increased, but levelled out as it got near 10 epochs.

GPT-2 is a pre-made model, so we did not have as much flexibility in terms of model shape or hyperparameters compared to LSTM. Therefore, there was a relatively low amount of experimentation necessary compared to our LSTM model. There are a few variants of the GPT-2 pre-trained model; we chose to use the relatively small "124M" variant as it had the most optimized training speed in exchange for accuracy. As we had limited time to learn and generate the text, we decided to use this variant as it was the smallest one, yet it still provided us with plenty of learning in order to produce text. One challenge that we expected was having to retrain the model each time we opened a new instance of the program, but GPT-2 had an in-built functionality called checkpoints. A checkpoint folder was generated after training the model; if it is saved, it can be used again to generate more text later. This made the problem of reusing the same trained model to be trivial.

The hyperparameters in the GPT-2 model were: tempera-

ture, generation length and a prompt. Temperature is a float value, usually from 0 to 1.5 that represents how random or volatile the generated text will be. Generation length is an integer which represents how many words that the model should generate. Prompt represents a string that will be some sort of "topic" used by the model to base it's generated text on.

As can be seen in the following graph, it appears that a temperature of about 0.8 can be considered as optimal for GPT-2 generation given our evidence as it will give a high value for both the character n-gram F-score and the word n-gram F-score. Thus, we used a temperature of 0.8. As with LSTM, we avoided low temperatures since they occasionally generated repetitive text.

In order to match the LSTM model so that we could compare the two models with the same amount of information, we chose 5 words as the prompt length, and we used the length of the original tweet that we used as the prompt for our generation length.

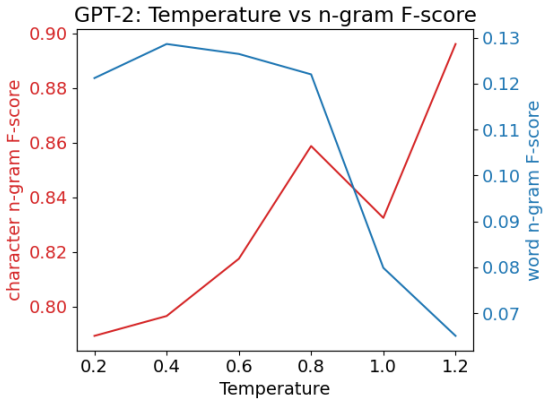

Figure 7: F-score for generated text vs temperature for the GPT-2 model. The F-scores using characters as n-grams is the graph in red, with the left y-axis scale. The F-scores using words as n-grams is the blue graph, with the right y-axis scale. As with LSTM, the character n-gram F-score tended to increase as temperature increased, while the word n-gram F-score tended to decrease.

Performance

As described in the "Evaluation Methods" section above, we evaluated our models based on two metrics: grammatical correctness, and style transfer.

For grammatical correctness, we used a human evaluation. We asked 12 people to evaluate our models. In total, 48 randomly chosen tweets generated by each of the LSTM and GPT-2 models were evaluated for grammatical correctness. As hypothesised, GPT-2 performed better than the LSTM model in both datasets on grammatical correctness. This can be attributed to the fact that GPT-2 is a far larger network than our LSTM's, and has been trained on much more data. GPT-2 achieved an average score of 6.394/10, while LSTM had an average score of 3.789/10 (Tables 1, 2). Thus, it has

	Avg Score (1-10)	Std Dev
LSTM Style Transfer	5.428	1.554
LSTM Grammar	3.736	1.627
GPT-2 Style Transfer	6.285	3.074
GPT-2 Grammar	6.210	2.043

Table 1: Human Evaluation based on the Trump dataset. 12 participants were asked to rate each type of generated tweet (LSTM, GPT-2) on style (conforming to what the participant expects from Trump) and the grammar of the tweet (is it legible?)

	Avg Score (1-10)	Std Dev
LSTM Style Transfer	5.714	1.637
LSTM Grammar	3.842	1.641
GPT-2 Style Transfer	6.857	2.381
GPT-2 Grammar	6.578	2.268

Table 2: Human Evaluation based on the News dataset. 12 random participants were asked to rate each type of generated tweet (LSTM, GPT2) on style (conforming to what the participant expects from a news outlet) and the grammar of the tweet (is it legible?)

had much more data to get a better grasp of English grammar rules. It is very easy to explain the ratings by inspecting a few example tweets from each model:

GPT-2 (trained on news tweets): *What's the secret to writing American books for President Trump? Read from the lead author of "Making America Great Again: Blueprint for the 2020 Food Revolution"*

GPT-2 (trained on Trump tweets): *This is good news, it is monumental. In a Republican Congress we should be focused on giving the American people the best healthcare policy they can afford. And to guarantee that every American has the best fixed healthcare option. And I won.*

LSTM (trained on news tweets): *What's the one thing you came to for the time of people who wear masks get infected with COVID-19. In it did not even attempt to figure out what percentage of people who wear a mask.*

LSTM (trained on Trump tweets): *Doing a GREAT job in Arizona win. No we need not pushed the Russian story as an excuse for running a terrible campaign. Millions spent on new fragrance and over China.*

In terms of style transfer, GPT-2 also outperformed LSTM, achieving an average rating of 6.571/10 compared to LSTM's 5.571/10. However, the margin is far smaller when compared to grammar rating. This indicates that the LSTM network is far better at capturing the context of the dataset relative to the the underlying language grammar-which is expected because the LSTM is only trained on contexts upon which it is graded upon (Trump, News tweets) whereas GPT2 has been trained on a much more varied dataset.

For our automatic evaluation methods of style transfer, we used an F-score metric using both characters and words as

	Avg char F-score	Avg word F-score
LSTM Trump	0.920	0.0778
LSTM News	0.929	0.0889
GPT-2 Trump	0.873	0.0746
GPT-2 News	0.871	0.0883

Table 3: F-scores for both models, which was used to measure style transfer. We measured all generated tweets using both characters (left column) and words (right column) as n-grams in the F-score algorithm, then averaged them over the model (LSTM and GPT-2) and training set (Trump and News).

n-grams (the details of the F-score metric were described in the "Evaluation Methods" section). We used a 5-word prompt from a real tweet from the target user, and generated tweets using both models with that prompt, and measured the F-scores for all subsequent words (including the first 5 words in the measurement would not make sense because they would always match). For example, for the real tweet was "A B C D E F G", and our model generated "A B C D E X Y Z", we computed the F-score using "F G" as the "correct" text and "X Y Z" as the generated text. The results of these evaluations are summarized in Table 3. Surprisingly, the F-scores for LSTM were better at both the character and word level than GPT-2. We suspect that this is because the LSTM was only able to generate words that were in the training set, meaning that its vocabulary is necessarily very similar to the actual Twitter user. This result did not align with our human-rated style transfer ratings. We suspect that this is because, although we explicitly asked people to ignore any grammar mistakes, it is hard to evaluate similarity of styles of two texts if one of them does not make grammatical sense, so the low grammatical correctness of the LSTM model influenced peoples' style transfer ratings. Additionally, the F-scores for characters are consistently significantly higher for character n-grams than for word n-grams. This is expected because of the domain size of characters vs words. There is a much larger selection of words that can be generated than characters, therefore the likelihood for our model to generate a single word that is also in the real tweet is much lower than the likelihood for it to generate a character that also exists in the real tweet.

Through working on this project, our group learned a lot about the process of building a machine learning neural network.

We learned that the data preprocessing step may require significant work and thought. In our case, we had to look at many different sources for large enough datasets, we had to consider which tweets to keep by filtering retweets and truncated tweets, had to modify capitalization on tweets so that our model could treat different capitalizations of the same word as the same, and we had to vectorize it to matrices of one-hot encoded vectors to represent words.

We learned that, when creating a neural network, a significant amount of time is spent on fine-tuning hyperparameters rather than building the initial model itself, especially because training takes so long, so checking the results of a minor change in the hyperparameters takes a long time.

Additionally, we learned that models trained on massive amounts of data will likely perform better in the context of Natural Language Generation (our GPT-2 model, which was a total of 2GB, performed much better than our 35MB LSTM models), and we think this may generalize to other applications of machine learning as well.

We also learned about how even clear steps and guidelines are unable to execute as planed. For example, we did not know of the twitter api limiting us to the most recent 3000 tweets of any user, which forced us to find different, third party datasets (ie, Kaggle dataset on Trump). This in turn changed our reprocessing steps to account for the different datasets peculiarities (ie, truncated tweets) and also shrunk our news dataset from 4.1M to 25k tweets.

Another important lesson is the importance of viewing of partial results when training. Instead of waiting 5 hours for 10 epochs to be finished training, far faster iteration can be achieved by checking the model quality every epoch (by generating tweets) and confirm that our model structure and hyperparameters are on the right track.

To view the source code for our project, please follow the link in the footnote. [1]

[1] https://github.com/Da-Pen/CS486-twitter-bot

References

[Case 2020] Case, K. 2020. Gpt-2 ai poetry generation: Writing like donne.

[Celikyilmaz, Clark, and Gao 2020] Celikyilmaz, A.; Clark, E.; and Gao, J. 2020. Evaluation of text generation: A survey. *arXiv preprint arXiv:2006.14799*.

[Devlin et al. 2018] Devlin, J.; Chang, M.-W.; Lee, K.; and Toutanova, K. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

[Han, Wu, and Niu 2017] Han, M.; Wu, O.; and Niu, Z. 2017. Unsupervised automatic text style transfer using lstm. In *National CCF Conference on Natural Language Processing and Chinese Computing*, 281–292. Springer.

[Liu and Liu 2019] Liu, D., and Liu, G. 2019. A transformer-based variational autoencoder for sentence generation. In *2019 International Joint Conference on Neural Networks (IJCNN)*, 1–7. IEEE.

[McCarthy and Jarvis 2010] McCarthy, P. M., and Jarvis, S. 2010. Mtld, vocd-d, and hd-d: A validation study of sophisticated approaches to lexical diversity assessment. *Behavior research methods* 42(2):381–392.

[Potash, Romanov, and Rumshisky 2015] Potash, P.; Romanov, A.; and Rumshisky, A. 2015. Ghostwriter: Using an lstm for automatic rap lyric generation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 1919–1924.

[Radford et al. 2019] Radford, A.; Wu, J.; Child, R.; Luan, D.; Amodei, D.; and Sutskever, I. 2019. Language models are unsupervised multitask learners.

[Sutskever, Martens, and Hinton 2011] Sutskever, I.; Martens, J.; and Hinton, G. E. 2011. Generating text with recurrent neural networks. In *ICML*.

[Tikhonov and Yamshchikov 2018] Tikhonov, A., and Yamshchikov, I. P. 2018. Guess who? multilingual approach for the automated generation of author-stylized poetry. In *2018 IEEE Spoken Language Technology Workshop (SLT)*, 787–794. IEEE.

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

[Wang et al. 2019] Wang, Y.; Wu, Y.; Mou, L.; Li, Z.; and Chao, W. 2019. Harnessing pre-trained neural networks with rules for formality style transfer. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 3564–3569.

[Wen et al. 2015] Wen, T.-H.; Gasic, M.; Mrkšić, N.; Su, P.-H.; Vandyke, D.; and Young, S. 2015. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 1711–1721.