CIS 520 Fall 2019, Project Report

My Beautiful Deep Neural Network: Applying Deep Learning Methods for Kanye West Lyric Generation

Team Members:

- Shirali Shah; Email: `shirali@seas.upenn.edu`
- Dhruv Iyer; Email: `diyer99@seas.upenn.edu`
- Alex Zhao; Email: `alexzhao@seas.upenn.edu`

Abstract

This project aims to explore lyric generation through Machine Learning by analyzing various generative Machine Learning models. The scope is limited to Hip-Hop lyrics, specifically from songs by the artist Kanye West. Some of the models we experiment with include one-gram, two-gram, three-gram, and four-gram Markov Models as well as two architectures based on LSTMs. We use Decision Trees to establish a baseline for comparison that is, in theory, naive without relying on uniform random sampling. In this report, we include our evaluations of the performance of all of these models in generating Kanye West Lyrics. We also include comparisons of their performances and discuss why some models and hyperparameters performed better than others. We ultimately find that the architecture of an LSTM network previously applied to predict the lyrics of another hip-hop artist, Drake, was the most optimal structure, showing the highest one-character prediction accuracy and lowest L_2 loss over the testing set.

1 Motivation

We seek to identify the best deep learning techniques to generate hip-hop lyrics. As discussed in section 2, recurrent ML methods have already been applied to text generation, but applications to hip-hop are understudied. Despite this, we believe hip-hop music tends to have more explicitly recognizable patterns in its lyrics, due to strict syllabic and rhyming requirements, which could make text prediction in this domain especially easy to learn. Furthermore, it would be interesting to see if a character-level model can pick up on the semantic meaning of hip-hop lyrics, as often the words artists choose are selected both for their rhymes and the subtle, emotional overtures they contribute to the lyric. At the same time, this meaning must be conveyed plainly and quickly, unlike in poems and novels, which might be a convenient simplification for ML models. Ultimately, we hope this study will provide insight into two key questions. First, is a character-level model, with current RNN architectures, sufficient to reasonably imitate a particular hip-hop artist's style? And second, can an artist convincingly use deep learning to generate lyrics during periods of writers block?

Assuming the answer to both of the above questions is yes, the next logical progression in this research would be to attempt to generate background tracks and synchronize it to the lyrics, however this is far outside the scope of this project. Still, for the interested, this piece explains why machine learning can be used in music and lyrics generation.

2 Related Work

The lyric generation problem has already been studied in several domains, including hip hop. For example, Ruslan Nikoleav did some work applying LSTM character-level models to generate Drake lyrics. Work has even been done specifically on trying to generate Kanye West lyrics. Robbie Barrat did this rather impressively by applying LSTMs and Markovify. However, Barrat's work was successful due in part to his placing strict guidelines on rhyming and syllable count, as well as learning a word-level model. We seek to study a harder problem of using a character-level model with no explicit guidance, making our work much more

generalizable.

In terms of the success of LSTM's in other domains, they have tremendous promise in many tasks, ranging from generating text and handwritten digits (Graves 2013) to speech recognition (Graves Navdeep Jaitly, 2014), image captions (Vinyals, Toshev, Bengio, Erhan, 2014), captioning videos (Venugopalan et al., n.d.), and many other applications. Deep LSTM architectures, especially bi-directional models have achieved incredible state of the art performance, especially on language model tasks, as demonstrated by GPT-2 (Radford et. al, 2019) and BERT (Devlin et. al, 2018), to name just two examples.

Related to lyric generation, a recent paper (Yu 2019) explores melody generation using a conditional Long Short-Term Memory - Generative Adversarial Network, using a deep LSTM generator and a deep LSTM discriminator, achieving impressive results. Indeed, Generative Adversarial Networks have achieved scarily impressive results in computer vision, natural language processing, time series synthesis, and more (Wang et. al 2019); Google's Magenta demonstrates incredibly powerful tools immediately at the musicians fingertips, from piano music generation (Hawthorne et al, 2018), to adversarial neural audio synthesis (Engel et. al, 2019). An increasing trend is the use of attention in neural networks to improve performance have shown promise in various domains, including machine translation (Luong et. al 2015).

3 Data Set

Here is the dataset of Kanye lyrics that we plan to use for lyric generation. It is a clean dataset, as it consists of pure text lyrics with no additional annotations. This is in contrast to, for example, this dataset from MetroLyrics, which would require significant preprocessing to be usable. On further inspection, we found the dataset to also be of a sufficient size and variety to learn a character level model, consisting of 6,191 lines of Kanye West verses. Following Nikoleav's formulation as a baseline, we decided our models should use the previous 20 characters to predict the next character of the verse. Slicing our dataset using a sliding window that does not cross between lines, this gave us 253,768 sequences, each consisting of 21 characters (20 input and one output). For our purposes, we partition this dataset into 162,411 training sequences, 40,603 validation sequences, and 56,944 testing sequences. To preprocess the data, we first lowercase all of the input and remove any newline characters. These simplifications were made to reduce the dimensionality of the data, as we did not feel they were essential for our proof-of-concept. More advanced work could reintroduce newline character prediction, but the effects of including newlines in the dataset are included in our analysis section. From there, we one-hot encoded every character in each sequence, leaving us with an input shape of $20 \times 69 \times 1$, as there were 69 unique characters in the corpus after the preprocessing. One of the limitations of using this dataset, we will note, is that we will only be able to produce Kanye West lyrics when there is more lyrical diversity among different hip-hop artists. We discuss this further in our analysis section.

Below is a frequency diagram of the characters in our vocabulary, as they are seen in the output set.

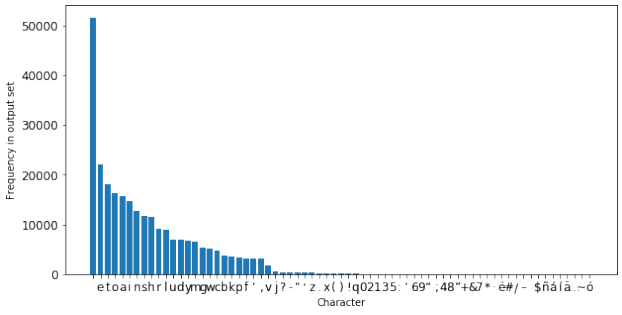

Figure 1: Character frequency plot

One interesting quirk of this data is that certain characters, such as the blank space and letters "e" and "t" appear significantly more frequently than others, resembling a Zipf distribution. This is also a classic problem in machine learning called unbalanced data. Therefore, overly simplistic or underfit models may be

tempted to simply predict one of the most commonly occurring characters. None of the models we developed were susceptible to this behavior, but in the event that they were, we likely would have had to downsample these characters in our training set to even out this property.

4 Problem Formulation

As discussed above, we will formulate this problem as follows:

> Given p characters of a lyric, predict the next character.

We believe $p = 20$ is reasonable, given previous work in this domain. One assumption we will introduce in this problem formulation is that in a sequence of characters, denoted by random variables X_i, so $X_1, X_2, ..., X_i, ..., X_{i+20}, X_{i+21}, ..., X_n$ that $(X_{i+1} \perp\!\!\!\perp X_1..X_{i-1}) \mid X_i..X_{i+20}$, or in other words, any character is conditionally independnt of the character that came before 20 characters before it. This is similar to the assumption made in Hidden Markov Models, but is critical for this problem formulation.

For the purposes of evaluation, we will study two metrics. First, we are interested in the one-character prediction accuracy. This is simply the proportion of samples in the testing set where our model would have correctly predicted the next character under argmax. This is most closely aligned with the problem formulation. Second, we will calculate the L_2 loss of our models to get a more holistic sense of the "goodness" of each. Ideally, the perfect model will predict 0 for every character except the correct one, whose value is 1.

For training our LSTM, our loss function of choice will be categorical cross-entropy, as this is the most commonly selected loss function in this domain. Similarly, we will use the RMS optimizer to perform our updates. We did not find feature engineering particularly relevant to this problem, and go into detail about our specific models int he next section.

5 Methods

In this section, we will describe the architectures of our three classes of models

5.1 Simple Markov Models (Baseline)

We must start with a baseline model for the lyrics generation, so we have chosen a Markov Model to be our base model. This is a good baseline because it is the simplest way to predict the next character using the previous characters using transition probabilities. This is just slightly better than choosing a character uniformly at random from the alphabet for each prediction because this model takes into account the prefix before the character and chooses the next character using probabilities determined by the frequency with which each character has been seen to follow that prefix in the training text.

We implemented the Markov Model by hand because it is such a simplified model. Additionally, because Markov Models are used for such a range of problems, there is so much customization required to meet the needs of the problem we are addressing. We trained the model by taking the training data of lyrics and constructing a frequency structure for it. We wanted the model to learn sequences of length $p + 1$ and note the frequency with which they appear in the training set to determine the transition probabilities from a p-length sequence to the next character. We used a structure that resembles a trie because we want to be able to map character sequences of some fixed length p to a transition probability for each of possible next characters. Some of the these character sequences have shared prefixes, which is why a trie-like structure would be easy to iterate over and require less memory. This frequency trie, stored as nested dictionaries, is used to get the transition probabilities between the preceding p characters and the next single character when feeding in the test data. These transition probabilities is what is used to determine the next single character. If a p-lengthed character sequence has not been seen before in the training data, the model selects a random character because all characters have a transition probability of 0 of following the p-lengthed sequence. When generating multiple characters, the model keeps a sliding window of length p and repeats the single-character prediction process until termination.

5.2 Decision Trees

We now seek to improve on the simple Markov model. One of the limitations of the baseline model is that it assumes an overly simplistic probability distribution over the possible next characters. It also implicitly gives equal weight to every character position, even though we would expect characters near the end of the sequence to be more important in determining the correct output. To address these limitations, we explore a model that has not traditionally been associated with text generation: decision trees. Our rationale for using decision trees is simple: we believe that a decision tree will be able to model a more complex probability distribution and learn and encode the elevated importance of characters near the end of the input sequence in a way that Markov models could not.

To implement our decision tree, we simply use the `tree` package from `scikitlearn` and fit it to our dataset. It is important to note that from the perspective of a decision tree, our sequence of characters is simply a vector with 20 features, one for each character in the sequence (we do not need to worry about one-hot encoding for decision trees, see LSTM section below), and there is no explicit temporal linkage between the states. Furthermore, the decision tree is not truly viewing the problem as text generation. Rather, the tree is attempting to classify the given input into one of 69 classes, one for each character, where the mapping between each training input and its corresponding output (label) is simply the next character in that training sequence. Essentially, our decision tree approach is really a transformation of a text-generation problem into a classification task. We discuss some of the strengths and weaknesses of this approach in our conclusion.

5.3 LSTMs

The above models were nonparametric and represented simplistic approaches to text generation. Our next approaches tend to look at the lyrical generation problem more semantically. Long Short-Term Memory cells are useful because they are an advanced form of Recurrent Neural Networks that solves many of the problems of Recurrent MLPs. We believe that LSTMs hidden state can be used to better encode information relevant to lyric generation, for example, a representation of the current syllabic rhyme scheme. Their increased complexity could lead them to show better results in this task. We will explore two LSTM architectures: one based on related research in this field and another with our modification.

One interesting point we should mention is that the input to our LSTM models must be one-hot encoded. Suppose it were not, then we might need to be concerned about enforcing scale invariance. In actuality, a "z" character should not have a greater magnitude than an "a" character, so we should not introduce that behavior into our models. By one-hot encoding the data, as Nikolaev did, we remove this dependency. This did not matter for decision trees or markov models because they are scale-invariant.

5.3.1 Drake LSTM

Our first LSTM architecture allows us to also explore the question of whether different musical styles can be encoded with the same architecture. We base this model off of work done by Ruslan Nikoleav to generate Drake lyrics using an LSTM. This architecture features a single LSTM layer with a hidden state size of 128, trained using RMSProp with learning rate 0.01 and categorical crossentropy loss. The Keras library provided an easy to use LSTM and training function, and we use this in our notebook (as did Nikolaev).

5.3.2 Modified Drake LSTM

After testing the Drake LSTM on our dataset, we were intrigued by the simplicity of the model, and wanted to explore how modifications to the network's architecture might influence the success of the model. We figured that increasing the model complexity and exploring different bias-variance tradeoffs might be worthwhile. In our modified architecture, we include a second LSTM layer with 256 hidden units that is used to encode the raw input. After LSTM-256 comes an LSTM-128 layer, then a final dense layer that results in the output. Our intuition behind this architecture is that the LSTM-256 layer should be able to encode the same information as the original LSTM-128 layer, but with the additional states, it may be able to track more meta information about the input sequence. The LSTM-128 layer is used to compress that information sequentially, so that the resulting network should be able to track changes in the hidden state over time. We

also believe that since the original Drake LSTM is a subset of this model, this architecture should perform better than the Drake LSTM. Though we are concerned about overfitting, we believe our usage of early stopping based on validation loss will combat this.

6 Experiments and Results

6.1 Configuring the Markov Model

Initially, for the Markov Model we tried to use $p = 20$ characters to predict the next character. After training on $p = 20$, we ran the model on the test data using $p = 20$ resulting in:

One Char Pred. Acc.	Mean L_2 Loss
0.996	0.203

Due to the poor performance of the model as given by the high average L_2 loss over all the samples in the test set, we decided to try different values of p to find a good configuration for this Markov model within the realm of generating Kanye lyrics.

Figures 2 and 3 show the model performance over different values for p: $p = [1, 5, 10, 15, 20]$

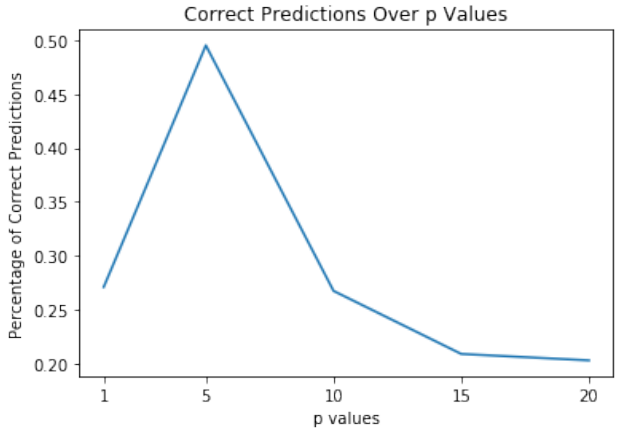

Figure 2: Markov Model Accuracy plot over p Values

Figure 2 the percentage of correct predictions over the test set for each of the p values.

Figure 3 the average L_2 loss over all the samples in the test set for each of the p values.

It is clear from Figures 2 and 3 that the higher values for p result in a worse performance; however having a p value that is too small also results in a terrible model performance as shown by $p = 1$. From these graphs we can conclude that having $p = 5$ is the best parameter for this model, so we use this as our baseline model.

6.2 LSTM Training Results

To train our LSTMs, we used early stopping on the validation loss (90/10 train/val split) and halted training when the validation loss did not chance significantly for two epochs. We used RMSProp with a learning rate of 0.01 because that optimizer and learning rate had seen success in Drake LSTM. Below are the training results for our LSTM networks

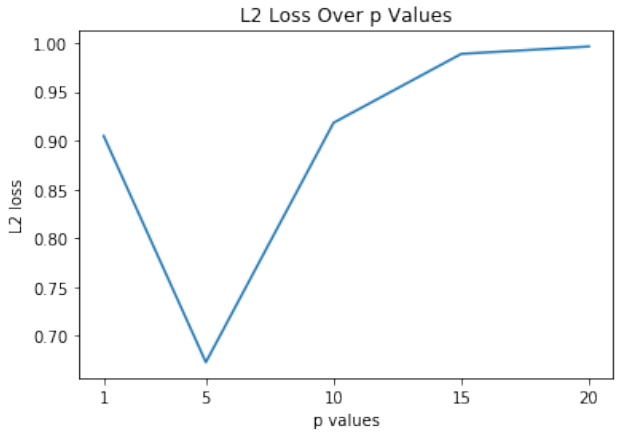

Figure 3: Markov Model L_2 plot over p Values

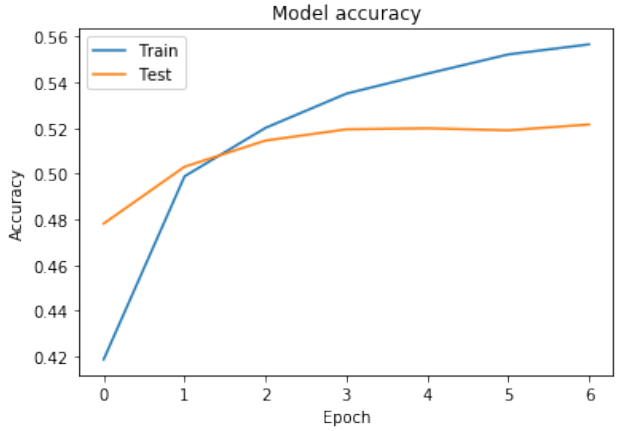

Figure 4: Drake LSTM Accuracy train/validation plot

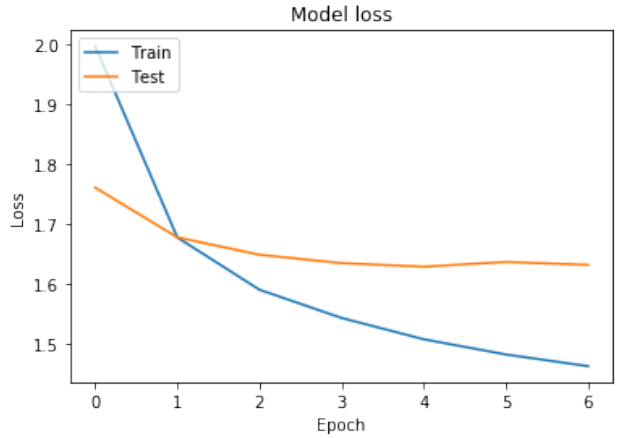

Figure 5: Drake LSTM Loss train/validation plot

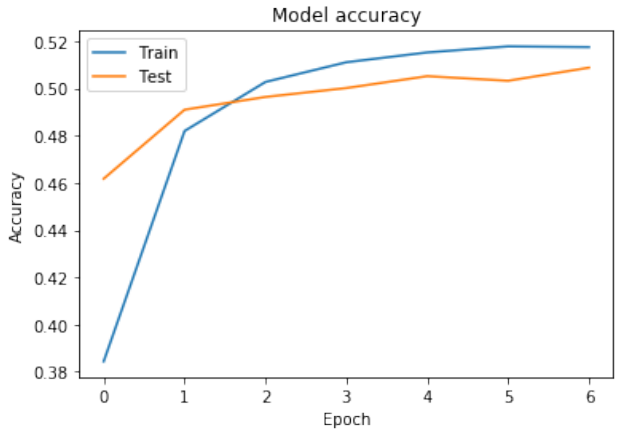

Figure 6: Modified Drake LSTM Accuracy train/validation plot

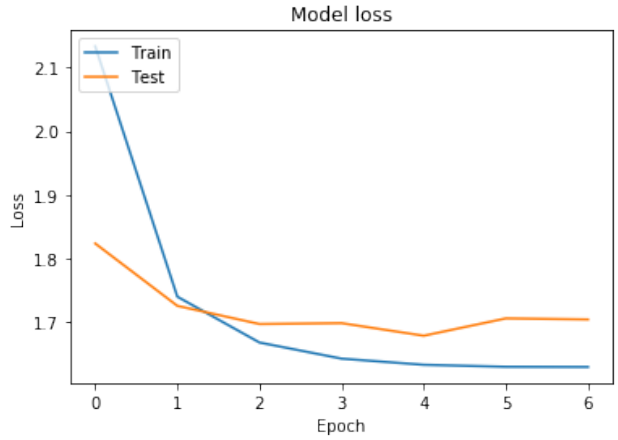

Figure 7: Modified Drake LSTM Loss train/validation plot

6.3 Model Evaluation Procedures

To evaluate our models, we calculated two metrics. First, we studied the one-character prediction accuracy. This is simply the number of times in our test set (about 20% of the total sequences available in the corpus) that the model, when seeded with the input vector, generated the correct character as its first prediction. This evaluation framework is useful because it gets to the heart of what our models are trying to predict, which is simply the next sample in a sequence generation task. Some may argue that for a text generation task, a one-character prediction accuracy is not semantically correct, as a longer forecast horizon would be better representative of a song generation task. However, we believe that the one-character prediction metric is sufficient for us to conclude that some models are more appropriate for this task than other. Furthermore, given the assumption that the next character in any sequence is conditionally independent of the characters before that sequence, we should be able to generalize the one-character prediction accuracy given the framework of our problem.

The other metric that we calculate is the L_2 loss between the output and the target probability vectors (the target essentially being one-hot encoded). The L_2 loss metric is important to know because it gives an indication of how confident each model is in the right answer. To calculate the one character prediction accuracy, we do not use stochastic sampling over the output probability distribution, as this would make the results somewhat non-deterministic. In lieu of that, our L_2 loss tells us whether a model was "hedging" its bets with multiple likely characters in the corpus. While the L_2 loss metric might not be useful when comparing models with significantly different one-char prediction accuracies, it can be relevant when comparing models where that statistic is similar.

6.4 Model Performances

This table shows the one character prediction accuracy and mean L_2 loss for all of our models

Architecture	One Char Pred. Acc.	Mean L_2 Loss
Markov 1	0.271	0.905
Markov 5	0.495	0.673
Markov 10	0.267	0.918
Markov 15	0.209	0.989
Markov 20	0.996	0.203
Decision Tree	0.430	250
Drake LSTM	0.515	0.662
Modified Drake LSTM	0.503	0.667

7 Conclusion and Discussion

Ultimately, we found that the existing Drake LSTM architecture was the superior model for generating Kanye lyrics. This has interesting implications for several reasons. First, it shows that two rappers with extremely distinct styles were able to be modeled by the same underlying architecture, which shows the versatility of LSTMs in the text generation task. Second, the Drake LSTM performed marginally better than the Modified Drake LSTM, even though the latter model was significantly more complex.

Decision trees surprisingly performed significantly worse than the best Markov model, despite our hypothesis that Markov models would provide an accurate and naive baseline. This indicates that the sequential nature that is built-in to the Markov model was more important than the robust probability distribution that was generated by the decision tree. The decision tree's significantly high L_2 loss is also concerning, because it indicates that in many cases, the model was extremely confident in the wrong prediction. In any case, we point out that our usage of decision trees would not scale well with other forecast horizons, as the number of potential "classes" would grow exponentially as a power of the vocabulary size.

The fact that LSTM models performed significantly better than the Markov model and decision tree is indicative that the hidden state structure is at the very least moderately useful for this task. Interestingly, we observed that excessively small or large Markov models tended to underfit and overfit the data respectively, but Markov5 performed nearly as well as the LSTM models. However, LSTMs are likely much more practical because, unlike Markov models, they do not require a large memory bank to store transition probabilities (and thus would scale better in a real world setting). Still, the fact that a model which only functionally relies on the previous 5 characters to predict the next character beat a model that ostensibly uses the whole input sequence tells us that our problem formulation may not have been the optimal compliment for our use of LSTMs.s

Rather unfortunately, no model we studied was able to reach a prediction accuracy much above 50%. This could indicate that a character-level model is insufficient to successfully predict lyrics, or that our prediction horizon did not offer suitable input.

7.1 Interpretations

Below, we have included several samples of lyrics that were generated from each model. This provides us with a basis for comparison.

Markov Model (p=5):

> ndon bloke, before h

Markov Model (p=5):

> ore hit hell how could questions running order i couldn't rap for my whole time you's, that your

Decision Tree seed:

> And they gon' hate i

Decision Tree output:

> And they gon' hate it and i'm his favorite i can't deny it, i'm a straight rider but when we get together be electric slidin' grandma, get 'em shook up aw naw

Drake LSTM seed:

> goin' coupe, I'm goi

Drake LSTM output:

> goin' coupe, I'm going on the day the past the past so i told the beat my man and the class

Modified Drake LSTM seed:

> to how I got here in

Modified Drake LSTM output:

> to how I got here in the top she said i got the sh*t the way the famous and that the best passion

In all these models, we can see that most of the lyrics it generates are real words, and some sequences within the texts are grammatically coherent, but these lyrics as a whole do not form sensible sentences, indicating that our models were not able to learn the semantic meaning of rap lyrics. The generated lyrics do preserve the slang the models find in the training data. Some of the sequences work for hip-hop because not everything in hip-hop requires complete sentences, but the parts of speech must be in a logical sequence . This tells us that, while our problem of generating Kanye lyrics may be easier than the conventional text generation problem, the problem structure requires us to look beyond lyric generation at just the character level. Doing so, ensures that we generate real words, but it does not attempt to place these words in a sensible ordering. What this tells us is that an analysis of words and parts-of-speech patterns would be needed in order to generate potential Kanye lyrics.

7.2 Process

We learned how significantly cleaning up and pre-processing the data can improve the performances of the models. Initially we had a larger vocab size by using every character found in the training data, unmodified. We tried reducing the character vocabulary by removing uppercase characters by converting them to lowercase as well as replacing new line characters with spaces. Reducing the vocabulary majorly improved the performances of the model because it removed unnecessary complexity. Additionally, we initially treated each line of a song as an independent sentence, preventing our p characters from crossing lines. We then tried to treat each line as a continuation of the line preceding it, so that there was crossing over lines for the p-lengthed character sequence used to predict the next character. This improved our model performances because lines in the same song make syntactical and semantic sense because they are logically a continuation of the thought before so this provided more data for the models to train on. However, the

lines that crossed from one song to another song did not make sense in sequence, but such occurrences are so rare in the data compared to pairs of lines within the same song.
We also learned that adding complexity to the models did not necessarily help model performance. We tried adding another layer to the Drake LSTM to make the Modified Drake LSTM, thinking this would improve performance because more information could be stored, but in actuality the performance of the modified Drake LSTM was worse than the original. When comparing the training and validation L_2 loss graph in Figure 5 of the Drake LSTM against that in Figure 7 of the modified Drake LSTM, we can see some overfitting. The training curve of the modified Drake LSTM drops quickly and begins to plateu quite early compared to that of the original Drake LSTM. This overfitting prevents the model from being able to generalize well on unseen data, explaining the worse performance. This shows that more complexity can lead to overfitting, which leads to poor generalization on new inputs.

We also tried to see how well a Decision Tree would perform. We expected it to perform better than the Markov model because it does not assume a uniform weight distribution across each of the p characters being considered when predicting the next character like the Markov model does. Instead, it gives higher importance to characters closer to the end of the p character sequence given that they would have the highest information gain. However, the Decision Tree seemed to perform worse than the Markov model, so this did not help, but this may be due to the fact that the Decision Tree does not take the order or characters into account whereas the Markov model does, so it appears that the order of the characters may have a higher information gain than their indices in the p long character sequence, explaining why the Markov model performed better because it uses the order of the p characters.

7.3 Future Research

Many future directions exist for this work, including both supervised and unsupervised techniques, both generative and non generative models. Future research areas can include: using Latent Dirichlet Allocation (LDA) and other topic modeling procedures such as Non-negative Mateix Factorization (NNMF) to create topic models for Kanye West Lyrics, or better yet pop songs in general (Blei 2012). LDA and other topic unsupervised techniques that can discover list of topics in Kanye West lyrics as well as Pop music as a whole. that is, LDA models each document, or in our case a song, as a list of topics. Each topic is a list of words that best describe the topic, each with some relevance measure, and these topics can be visualized tools such as pyLDAvis. Fitting a LDA model to the data gives us a topic-document matrix (what topics exist in each document) in addition to a term-topic (what words describe each topic), the first of which gives us a vector representation of each verse or song, depending how we slice up the original dataset. Along with other vectorization techniques such as Term document counts optionally weighted by TFIDF, or using retrained embeddings such as GloVe, Fasttext, Doc2Vec, we can also perform various other unsupervised analysis such as clustering other to find naturally occurring genres, and similar songs. Indeed, the vectorization of words and documents using appropriate embedding is often the first step to multiple additional forms analysis, and is an example of semi-supervised learning which combines pre-trained, supervised features with unsupervised techniques. Along with with sentiment analysis using pre-train models and thr attributes present in Spotify's song API we can use vectorizations of documents to build the beginnings of a music recommendation system.

Another exciting direction of research is to use fine-tuning and transfer learning to leverage state-of-the-art pre-trained models for this language generation task. Open AI's GPT-2 and the most recent BERT models by Google and competitors are bi-directional attention based LSTM architectures. These models can be fine-tuned with our own training data to provide state of the art generation results. Indeed, the results published by OpenAI on their blog and interactive example on talktotransformer.com are stunningly interpretable. One good starting point for these approaches is huggingface.com, which provides good instructions for how to fine tune state of the art models.

Moving beyond simple text generation and modeling documents with a generative model, there is also work being done to generate album art from lyrics, music videos from lyrics, as well as melodies from lyrics, most of which leverage LSTM in conjunction with adversarial generative networks. An interesting future direction is to use Google Deep Dream, LSTM lyric models, and to reverse engineering image captioning CNN models to Deep Dream like music videos and images. Indeed there is exciting work being done in this area, It relates closely to the idea of and relates closely to the idea of training multiple models with single objective functions for better performance and faster training, and remains a very exciting area of research.

Extending beyond model creation, it is also worth looking at model evaluation as well as feature importance. We can leverage the recently published LIME algorithm (local interpretable model agnostic explanations) to detect which part of Kanye West lyrics most lend style to Kanye West, identifying the "signature twists" of different artists. This can also be used in conjunction with a labeled corpus complete with artist names and other meta-data for a variety of interesting tasks. With the appropriate meta-data, we can perform classification tasks to predict the artist name from lyrics given text features, or predict any multi class or even continuous labels such as genre, valance, artist age, and target audience, lending itself to extensions in music recommendations systems and can indeed be quite profitable. The ability to learn particular artist styles also opens a whole can of possibilities in evaluating music authorship, and may have extensions in intellectual property, copyright infringement, and examining the uniqueness of current popular music. Another related natural extension is by using variational topic models (Wang et. al 2012) to examine the evolution of musical topics over time.

These future directions naturally arise from interesting topic domains in natural language processing, and remind us of the fact that proper vectorization and representation with feature engineering leads to more generalizable models and allows us to perform many interesting machine learning tasks. Amidst all of these possible future extensions, it is worth noting what are the foundational principals shared among all of these machine learning algorithms: With sufficient data, appropriate representation and feature engineering, and a sensible loss function, the possible future research directions and extensions are limitless (Domingos 2012).

A meta-comment about model evaluation is also appropriate and much needed here. Our data was a unlabeled corpus that contained only raw text, in a single txt file. What actually constitutes good lyric generation is extremely not obvious. In our paper, we have used two evaluation metrics:one character prediction accuracy based on the testing corpus, and L_2 loss of our models over the entire testing. What that explicitly means is that the built in assumption is Kanye West lyrics are the gold standard, and serves as the "right" target for evaluation. But how true is that? Perhaps the next frontier of machine learning is its role in create production, working alongside artists and musicians. This begs the question, how do we design future systems that also reward creativity, and how do we define a sensible lost function that captures the abstract notion of creativity? Currently, the parameter of "temperature" in LSTM's and generative models is closely related to the models' perceived expression of "creativity". This parameter specifies the degree of randomness, or high temperature search, that the machine learning algorithm will incorporate. As such, a natural future direction is to fine-tune these temperature parameters and research more appropriate targets for defining loss functions and evaluation metrics.

All in all, due to the scoping and time requirements, we chose to limit our work to a very concrete algorithms including Markov Models, multi layer perceptions, and LSTM's with minimal modifications. To extend this work into a PhD level thesis, with addition considerations for bias, topic models, sentiment, and ultimately creativity in recommendation systems or novel generative models indeed is not too much of a stretch.

Acknowledgments

We would like to thank Professor Lyle Ungar for his guidance and support, including invaluable help in scoping the project. We would also like to thank Ruslan Nicholaev, a data scientist writing for Toward Data Science, which provided valued inspiration for our examined architecture, as well as sample code which we built on to implement our LSTMs and parse our dataset. Lastly, we would like to thank Vicc Alexander for providing the dataset of Kanye lyrics which we used for this project.

8 References

Barrat, R. (2019). Google Colaboratory. Retrieved December 11, 2019, from Google.com website: https://colab.research.googl uWGZLABnN0

Robbie Barrat generating music with LSTMs and Markovify. Christoph Molnar. (2019, September 18). 5.7 Local Surrogate (LIME) — Interpretable Machine Learning. Retrieved from Github.io website: https://christophm.github.io/i ml-book/lime.html

Colah, C. (2015, August 27). Understanding LSTM Networks – colahâs blog. Retrieved November 11, 2019, from Github.io website: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Devlin, J., Chang, M. W., Lee, K., Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.Chicago BERT Language Model, 2874 citations

Engel, J., Agrawal, K. K., Chen, S., Gulrajani, I., Donahue, C., Roberts, A. (2019). Gansynth: Adversarial neural audio synthesis. arXiv preprint arXiv:1902.08710.

Graves, A. (2013). Generating sequences with recurrent neural networks. arXiv preprint arXiv:1308.0850.

Graves, A., Navdeep Jaitly. (2014). Towards End-To-End Speech Recognition with Recurrent Neural Networks. PMLR, 1764â1772. Retrieved from http://proceedings.mlr.press/v32/graves14.html

Hawthorne, C., Stasyuk, A., Roberts, A., Simon, I., Huang, C. Z. A., Dieleman, S., ... Eck, D. (2018). Enabling factorized piano music modeling and generation with the MAESTRO dataset. arXiv preprint arXiv:1810.12247.Chicago

Hitchings, P. (2018, August 23). Music and Machine Learning - ai.SensiLab. Retrieved December 11, 2019, from Monash.edu website: http://ai.sensilab.monash.edu/2018/08/23/Neural-Music/

Kaparthy, A. (2015, May 21). The Unreasonable Effectiveness of Recurrent Neural Networks. Retrieved April 28, 2019, from Github.io website: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Luong, M. T., Pham, H., Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. arXiv preprint arXiv:1508.04025.

GyanendraMishra. (2017, January). 380,000+ lyrics from MetroLyrics, Version 2. Retrieved November 10, 2019 from https://www.kaggle.com/gyani95/380000-lyrics-from-metrolyrics/. Metrolyrics Dataset

Nikolaev, R. (2018, April 9). Generating Drake Rap Lyrics using Language Models and LSTMs. Retrieved December 11, 2019, from Medium website: https://towardsdatascience.com/generating-drake-rap-lyrics-using-language-models-and-lstms-8725d71b1b12

Odena, A. (2019). Open Questions about Generative Adversarial Networks. Distill, 4(4). https://doi.org/10.23915/distill.0001

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I. (2019). Language models are unsupervised multitask learners. OpenAI Blog, 1(8). GPT-2 Language Model by Open AI

Ribeiro, M. T., Singh, S., Guestrin, C. (2016, August). Why should i trust you?: Explaining the predictions of any classifier. In Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining (pp. 1135-1144). ACM. LIME Model

Varoquaux, G., Buitinck, L., Louppe, G., Grisel, O., Pedregosa, F., Mueller, A. (2015). Scikit-learn. Get-

Mobile: Mobile Computing and Communications, 19(1), 29â33. https://doi.org/10.1145/2786984.2786995

Venugopalan, S., Rohrbach, M., Donahue, J., Mooney, R., Darrell, T., Saenko, K. (n.d.). Sequence to Sequence -Video to Text. Retrieved from https://arxiv.org/pdf/1505.00487.pdf

Vicc Alexander. (2016, September). Kanye West Rap Verses. Version 1. Retrieved November 10, 2019 from https://www.kaggle.com/viccalexander/kanyewestverses. Selected dataset

Vinyals, O., Toshev, A., Bengio, S., Erhan, D. (2014). Show and Tell: A Neural Image Caption Generator. Retrieved from arXiv.org website: https://arxiv.org/abs/1411.4555

Wang, C., Blei, D., Heckerman, D. (2012). Continuous time dynamic topic models. arXiv preprint arXiv:1206.3298.

Wang, Z., She, Q., Ward, T. E. (2019). Generative Adversarial Networks: A Survey and Taxonomy. arXiv preprint arXiv:1906.01529

Yu, Y., Canales, S. (2019). Conditional LSTM-GAN for Melody Generation from Lyrics. arXiv preprint arXiv:1908.05551.