Lil' Baughty - Generating Rap Lyrics

Christopher Hartman / Jared Schifrien / Sasha Weiss

Utilizing an LSTM model in the TensorFlow and Keras frameworks to synthesize rap lyrics

I. Introduction

While there has been a great focus on the applications of RNN and LSTMs in prose language, natural language processing has not been used nearly as significantly in the field of non-prose language such as musical lyric generation. Unlike text such as the Wall Street Journal or Wikipedia, lyrics are more heavily structured and are tied to attributes such as rhythm, melody, beat, and rhyme. Specifically, in this report we focus on the genre of rap, due to its especially structured flow and sentence structure. This poses a novel challenge for existing language models due to the context-rich and -dependent nature of rap lyrics beyond solely the preceding text. This work contributes also to a larger movement of synthetic music generation, including computer-enhancement or replacement of human vocals and auto-synthesis of lyrics. We believe that synthesizing lyrics has not only immediate academic and industrial application, but will also be endlessly entertaining: robot rap battles!

II. Related Work

In Mikolov's "Recurrent neural network based language model" it was first proposed to use a RNN for the application of natural language processing, jumpstarting the field and allowing for significant improvements over the commonly used N-gram models. An RNN allows for a much longer context memory to generate a probabilistic distribution for predicting words than only viewing the last N words. For our model specifically, this is incredibly necessary due to the context-rich nature of the text. Without longer context memory, the overall flow of a generated lyric and overall melody would be extremely inconsistent.

"Generating Sequences With Recurrent Neural Networks" has a major relevance to our paper since we use a RNN with a very similar architecture that uses LSTMs (Long Short-term Memory). LSTMs are used to stabilize the output of the model so that it is based on long-term input but with an increased focus on short-term context. While the paper focuses on prose text generation and handwriting synthesis, we sought to utilize the LSTM system (as opposed to a GRU) to better predict lyrics due to the context rich nature of lyrics. Furthermore, we have found a paper "A First Look at Music Composition using LSTM Recurrent Neural Networks" that first proposed the use of LSTMs in order to generate music but without the lyrical component. This gave us insight that a basic LSTM architecture is a viable choice for understanding beat and melody and can be applied to non-prose applications.

The off the shelf TensorFlow model that we originally used is based off of "Recurrent Neural Network Regularization" that applies dropout to the LSTM architecture in order to regularize the neural network. By using this method, we are able to use a much larger dataset without the worry of excessive overfitting.

III. Data Collection and processing

A. Lyric Scraping

Due to copyright laws, lyrics are not readily available in any dataset aside from bag-of-words data. While websites offer the lyrics of singular songs, we required a much larger corpus to accurately synthesize lyrics. To solve this we wrote a Python web scraper (Appendix A) that collected the lyrics of a given artist ID on Genius.com into a series of text files. After scraping Artist ID's 1-2500, we were left with 1911 total artists (some artist IDs led to a 404) and the lyrics of every song they either produced or were featured on. We found that Genius.com also often includes speech transcripts or other non-musical text, which we were able to clean based on the assumption that lyrics would contain bracketed annotations such as "[chorus]". After using a Bash script to remove duplicate songs and concatenate the text into one large text file, we were left with a 233.2 MB file, from which we removed punctuation and replaced newlines with an end of sentence token. We would eventually use a ~20 MB subset to train our models.

B. Cleaning

Since Genius.com allows users to submit lyrics with very little moderation, there are no grammar or spelling constraints on the lyrics of a song. Because of this, the dataset included an incredibly large vocabulary (50,000+ words) and a lot of what we labeled as "strange words" that only had made sense in the context of a song or specific verse. Furthermore, words such as "oh" often had multiple spellings in the dataset such as "ohh" or "ooh", leading to a falsely large vocabulary. To cut down on the vocab, we removed the lines of text that contain a word not found in the 25,000 most common words in the dataset. Finally, we qualitatively analyzed the top 25,000 words to ensure that they could still produce coherent lyrics.

IV. Model & Results

A. Initial Modeling with TensorFlow

TensorFlow is an open source software library that enables the use of machine learning models to perform most machine learning tasks. We used an "off-the-shelf" RNN that uses LSTMs to predict text in the Penn Treebank dataset, tuned to replicate the results from a recent paper on LSTMs (Zaremba et al., 2014). The model also employs a stacked LSTM in order to improve the results. Several configurations are available for the model, a test (very small), small, medium, and large configuration. The small configuration trained in about 12 hours on 15 MB of data, and most of our examples are generated using that model. This configuration has 2 layers and a hidden size of 200.

Since we cleaned our data very similarly to the Penn Treebank data, the model was easily adapted to use the lyrical data. The primary difference was the increased vocabulary of our data, (~10,000 for Penn

Treebank vs ~25,000 for ours), and the larger size of our dataset. This model encodes text by counting occurrences of each word, sorting the words by their frequency, and mapping each word in the sorted list to its index in the list. We had to optimize this process by saving and restoring the results in a python pickle file instead of recalculating them each time the model was run, since it took much longer on our larger dataset. We considered investigating other vector representations, but we found this system to work sufficiently well.

We heavily modified the system to generate text as opposed to solely predicting on the testing set. To do this, we added probability predictions to the outputs of the model by returning a softmax of the existing logits. we sample randomly from the probabilities, using a random multinomial sample from the set calculated by equation 1, where D is the data we sampled from, p is the probabilities output by the model given the previous word and state, and t is the temperature. We experimented with several temperatures for the sample, and found that values between .5 and 1 produced results which we found relatively coherent without being repetitive and predictable.

(1) $D = softmax(log(p)/t)$

Due to computational complexity, we trimmed the initial dataset down to a 15MB training set, a 3MB validation set, and a 1MB Testing set. Utilizing the small model hyperparameters tuned for PTB, we arrived at a 120 perplexity, while increasing the size of the model led to a perplexity of 100. Qualitatively, we found that there wasn't a significant difference between lyrics generated with the 120 perplexity model and the lyrics generated with the 100 perplexity model, likely because of the randomness introduced in the sampling.

In qualitative analysis, we found that the model excelled at generating long sets of lyrics that follow a consistent tone, but still failed oftentimes to create lyrics with any substantial meaning. It's possible that with more data, the generated lyrics would be more cohesive. We also found that the model does occasionally produce rhymes, but not nearly as often as actual rap lyrics.

B. Further Modeling with Keras

While the pre-built TensorFlow model described above generated lyrics well with only minor tuning, we were concerned about our ability to correctly implement more complex training modifications such as the incorporation of rhyme. Consequently, we rebuilt the word-level generation scheme described above using the Keras wrapper on TensorFlow, and the later stage of our work as well as future work will rely more heavily on this model.

C. Rhyming

We took several approaches in attempting to incorporate rhyming into our generated text. Unfortunately, rhyme is extremely complicated, and generally the approaches were unsuccessful. One of our first ideas was to implement a beam search while generating text, and optimize for words/lines that rhyme. This

suffered from two main weaknesses. First, even with a sufficiently large beam size, the model rarely generated a single rhyming line, so the beam search was optimizing over equally poor lines. Second, it's difficult to score the rhyme of a word, especially given that our model generates a lot of words that aren't in dictionaries like the CMU Pronunciation dictionary, and given that rappers are famous for their use of "slant rhymes" (word that don't technically rhyme, but sound similar enough, like "orange" and "door hinge") that wouldn't score using a standard rhyming metric. Our attempts to create a score included both a score based on whether or not words rhymed and also based simply on how often sounds were repeated. Both had significant weaknesses, and also tended to prioritize repeated words rather than genuine rhymes. We believe that further efforts with this method could be successful, but we would need to add words from our dataset to the pronunciation dictionary.

Besides using the CMU dictionary, we also experimented with building rhyme schemes using the Double Metaphone (DM) algorithm, which builds 0-4 letter pronunciation hashes of a word. When finding a rhyme for a word, we scored all other words in our vocabulary by how much their hashes overlapped with our target, starting from the back of the word. Unfortunately, DM is primarily designed to capture the high-level pronunciation structure of an English word, and consequently failed to capture the subtle semantics of rhyme at the end of words. For example, in the DM scheme all vowel sounds are assigned the character "A".

V. Conclusion and Future Work

A. Conclusion

From our initial results we can qualitatively observe that stacking the LSTM in multiple layers results in a much more coherent lyric generation. Due to the small size of our initial tests increasing the size of our dataset consistently leads to lower perplexity without overfitting, either due to the small size of the dataset or the regularization of the RNN.

The most significant takeaway from this experiment has been the power of LSTM's, and RNN's in general, to be adapted to different kinds of text. They are able to pick up the subtleties of different genres of text, and our model was able to generate text that is observably closer to rap lyrics than any other form of text. This is a powerful insight into the current state of natural language generation, and shows signs that the current models are applicable to a wide range of uses, not just simple tasks.

Furthermore, the idea of rhyming the last word of a sentence creates a struggle between the probabilistic distribution of what word fits in context and whether it rhymes. Rhyme is an interesting dilemma for a model like this, because words are only used to rhyme some of the time, meaning the model would have to learn both how and when to use a word to produce a rhyme, as well as how and when to use that word in standard contexts. The model clearly struggles between these two situations, often rhyming when it doesn't need to and throwing off the actual meaning of the lines, or failing to rhyme when it should.Overall we are pleased with how our model was able to handle the novel challenge of generating lyrics based on context-rich input and successfully generate entertaining rap lyrics.

B. Future Work

With the model we have right now, we would like to further increase the size of our training/validation/testing sets to obtain a larger corpus from 20MB to the full 100MB dataset we have scraped. Furthermore, we should be able to scrape for more artists from the 1,911 we have now to the ~20,000 on Genius.com. Thirdly, with more computational resources, we would like to test models with larger configuration sizes with more hidden layers and layer sizes. Additionally, with the model we have right now, we had wished we could train on different artists and subgenres of rap to learn what performs the best and how the model handles different forms of rap.

Our model currently uses flawed methods for creating rhyme structure; consequently, another big area of future work is improving and formalizing the structure of the rhyme synthesis. We currently use Double Metaphone and CMU dictionary approaches, which may be feasible in and of themselves with additional tuning. Projects such as http://www.afader.com/2012/06/11/pochemuchka.html indicate that other NLP techniques might be valuable in this space as well.

Finally, investigating other aspects of rap such as syllabic structure and metadata of the musical accompaniment may be interesting, as they could help capture more subtle aspects of rap without proving unfeasible in terms of data collection.

VI. References

D. Eck and J. Schmidhuber. A first look at music composition using lstm recurrent neural networks. Technical report, IDSIA USI-SUPSI Instituto Dalle Molle. (http://people.idsia.ch/~juergen/blues/IDSIA-07-02.pdf)

Graves, Alex. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013. (https://arxiv.org/pdf/1308.0850v5.pdf)

Mikolov, Tomas, Karafia ́t, Martin, Burget, Lukas, Cernocky`, Jan, and Khudanpur, Sanjeev. Recurrent neural network based language model. In *INTERSPEECH*, pp. 1045–1048, 2010. (http://www.fit.vutbr.cz/research/groups/speech/publi/2010/mikolov_interspeech2010_IS100722.pdf)

Zaremba, Wojciech, Sutskever, Ilya, Vinyals, Oriol. Recurrent Neural Network Regularization. CoRR, 2014. (https://arxiv.org/pdf/1409.2329.pdf)

Appendix A: Web Scraper

See https://github.com/sashaweiss/rap_bot/tree/master/scrape.

We queried pages of the form https://genius.com/artists/songs?for_artist_page=2, filling in the for_artist_page parameter with the numbers 1-2500 incrementally. From there, we pulled a link to each song by the artist given by for_artist_page, and extracted the lyrics. Fun fact: Genius doesn't block IPs when you hammer them from 10 EECS servers at once!

Having downloaded the lyrics to each song by each artist, we concatenated all of them into one file and cleaned it (as described above). Subsets of that file were then used for our experiments.

Appendix B: TensorFlow/Keras Code Addition

We implemented word-level LSTM generators using the following tutorials and examples:

- https://www.tensorflow.org/tutorials/recurrent
- https://github.com/fchollet/keras/blob/master/examples/lstm_text_generation.py

Our implementations can be found here:

- https://github.com/sashaweiss/rap_bot/tree/master/keras
- https://github.com/sashaweiss/rap_bot/tree/master/lstm

Appendix C: Selected Censored Generated Lyrics

“ i am a fighter
for my motivation
and i am brave i feel kinda fast
i feel like i feel theres someone to love me”

“if you is a b**** still you aint gone trust her
them streets is laughin son n****s pissed off
broke n**** and i forgot for life
i dont need to be a man like this for two months”

These two lyrics demonstrate the model’s ability to differentiate between vastly different tones. In the first lyric, the model is using positive words and has an optimistic tone. In the second lyric, the words are much more negative and the tone is darker overall.

“im not your p**** man
oh you n****
you a f***ing b****
you just a f***ing girl have had to lick this up”

This lyric demonstrates the difference between our dataset and a standard dataset. Explicit words are much more common in this data, and the model responds appropriately.

“cool about a refrigerator life
let me ask you when im walking through
im a psychopathic hustler nigga
imagine when it was cold”

The model seems to be stuck between the two meanings of “cool” here.

“yeah
rap futuristico
banged
tranne te tra me e te tranne te rap futuristico
tranne te tra me e te tranne te tranne te
tranne te tra me e te tranne te tranne te
tranne te tra me e te tranne te rap futuristico
tranne te tra me e te tranne te tranne te
tranne te rap futuristico
tranne te tra me e te tranne te rap futuristico
tranne te tra me e te tranne te rap futuristico

tranne te tra me e te tranne te tranne te
tranne te tra me e te tranne te rap futuristico
tranne te tra me e te tranne te tranne te
tranne te tra me e te tranne te rap futuristico
tranne te tra me e te tranne te tranne te”

The model fails here because the random sample produced the words “rap futuristico”, which only appears in one song in the dataset, which led the model to repeat the chorus of that song, a chorus which is primarily even more obscure words, trapping the model in the one song.