Generating Structured Text Using Multi-Stage Model for Specialized Applications

Bomi Kim and Paul Stott
University of California, Berkeley
{bomi.kim, paul_stott}@berkeley.edu

Abstract

Machine rap lyric generation poses a difficult problem due largely to the unpredictable patterns in word selection. We aim to analyze the songwriting process and solve the writer's block problem, which is given a number of lines of poetry, what is a suitable line to follow.

1 Introduction

Akin to poetry, rap has been produced at differing degrees of technicality, structure, and style. Both forms of communication take advantage of the phonological aspects of the language in which they are written and organize words in a way that conveys meaning on other dimensions apart from the prosaic or narratological dimension of most other forms of communication. Although both poetry and rap share this attention to the euphony of phonemes, rap does not traditionally follow any strict patterns that its ancestor espouses. While poetry has rhyming schemes, meter, and other formal structures, rap has looser constraints. This makes the generation of rap unique. The formalisms upon which a given rap is structured might not have a predictable cadence, nor does a body of work of a given artist follow predictable patterns as a poet like Shakespeare in his sonnets or plays.

Given the vastness of the problem space in rap lyric generation, we have scoped our work to the writer's block problem, that is, given *n* lines, what is a suitable line that will follow. Suitability, in this case, accounts for not only semantics but the dimensions of style encoded in the previous *n* lines. In order to produce the best fit for a generated line, an understanding of all the dimensions of style, structure, and technicality, specific to rap, need be operationally defined. Our process includes the process of encoding these dimensions for a corpora of rap artists' lyrics and then using that understanding to generate suitable lyrics given an input sequence of lines.

2 Related Work

Adjacent work has focused on incorporating strict structures and constraints in text generation. Of all the work in this field, we have focused on work in feature engineering in poetic forms and work that makes use of those features in a language model.

Approaches to encode poetic formalisms focus on the quantification of aesthetics, of which we are not concerned. *A Computational Analysis of Style, Affect, and Imagery in Contemporary Poetry* (Kao and Jurafsky, 2012) provides insights on identifiable features, ranging from diction (choice of words, word frequency, etc.), phonological devices (end rhymes, assonance, consonance, etc.) to imagery. Their work, however, does not focus on text generation. *DopeLearning, A Computational Approach to Rap Lyrics Generation* (Malmi et al., 2015), on the other hand, provide an approach to not only the definition and quantification of poetic features, but also the design of a language model for lyric generation. This method takes a novel approach in the generation process where the output is dependent on the computed features of the input. Extrinsically, the output lyrics are suboptimal and don't stand up to a human lyricist. Given their outcome, it suggests that performance could be improved given a more

robust analysis of poetic features and additional work in model design.

For work in language model design in lyric generation, *Automatically Generating Rhythmic Verse with Neural Networks* (Hopkins and Kiela, 2017) introduces intuitive methods. With rhyming being the most predominant poetic feature in rap, a Long-Short Term Memory network model that takes phonemic encoded words and returns a phonemic representation as output. Using the phonemic representation, searching an outcome space for an appropriate word simply requires matching a sequence of phonemes to a word. By stacking layers of models to incorporate semantic representation, the model encodes two main features of generating lyric or poetry: themes and poetic devices. Similarly, *Generating Chinese Ci with Designated Metrical Structure* (Zhang et al., 2019) introduces an approach in which a strict metrical structure is required for successful poetic generation. The structure of Chinese Ci consists of three rules concerning rhythm, tone, and rhyme respectively. Those features are encoded and then used to train a neural model to generate Ci that follows those rules. While the concepts from (Hopkins and Kiela, 2017) and (Zhang et al., 2019) can be useful in our project, rap doesn't have a strict structure but rather relies on other phonological qualities of language to produce euphony. For example, in Ci, most rhymes are end rhymes based on a limited set of vowel tones. (*"Every character was pronounced in one of five tones." Zhang et al., 2019, p.7460*) Rap, however, can have multiple rhymes with varying syllable lengths at unpredictable positions. Such patterns are difficult to encode because the window to search for rhyming phonemes isn't strictly determined. To address this issue, Herjee and Brown (2010) suggest a method of: 1. scoring potential rhymes using a probabilistic model based on phoneme frequencies and 2. characterizing rhyming style by artists. The method can play a great role in detecting imperfect / polysyllabic / internal rhymes that are difficult to detect using the aforementioned methods in the works discussed in this section. However, training the model is labor-intensive (about 28,000 lines of lyrics with 14,000 rhymed paris that were annotated by humans). While it might be an accurate way to obtain training data, other methods should be considered for the sake of efficiency.

Transformer based models have also been employed for text generation, although not strictly for lyric generation. These models provide robust encoding of style and have produced incredible results across many text generation tasks. *CTRL* (Keskar et al., 2019) is such an approach, allowing users to control particular aspects of the generated text at generation time using "control codes". This allows for genre-specific generation with a lighter training requirement. Given our task, complying with the requisite fine-grain control at generation time would result in a proliferation of context specific "control codes". This approach doesn't fit with our task since rap lyric generation is author dependent and doesn't necessarily represent a unifying poetic style of an identifiable group that could be represented in a code.

Finally, Watanabe et al. (2017) provides useful insights on a user interface for lyric generation called *LyriSys*. An interactive interface like *LyriSys* allows for the user's behavior to become feedback on which the model can train. Additionally, it suggests a simple text generating model that's based on trigram probabilities, constrained by theme and syllable count, allowing for thematically relevant text generation while enforcing poetic forms.

3 Data

Our dataset was compiled from data scraped from `genius.com`, a crowd-sourced lyric database that includes metadata about songs that allow for classification of rappers in a variety of dimensions. Our initial exploration dataset consists of lyrics from a set of 20 popular rappers, including all songs on each artists studio albums. Lyrics are in a plain text format with new line indicators and no other non-lyrical data.

A more complete corpus of rap lyrics are compiled for training the selected model. This corpus includes three hundreds English rap artists, spanning four decades and geographic locations. The data are structured as text files and blocks indicate a section of the song, either a verse, hook, chorus, refrain, etc.

4 Methodology

Inspired by (Hopkins and Kiela, 2017) and (Zhang et al., 2019), we developed a multi-layer encoder - decoder model. As discussed, the multi-layer model can incorporate two main categories that define rap, poetic devices and themes, in a more controllable way. This methodology is also beneficial to further work as it grounds its performance on deeper understanding of the structure.

In constructing the model, we use phonemes as a building block. English words can have different pronunciation for the same characters or for the same words, the information only phonemes can capture. We give each phoneme a unique vector value, allowing the model to learn the rhyming structure and semantics at the same time.

4.1 Phoneme as a building block

Our focus is to transcribe each token into a representation that allows for the comparison of sound sequences while still maintaining the semantic information. Our lyric dataset was transcribed to phonemes in ARPAbet using the The Carnegie Mellon University Pronouncing Dictionary. Lines and words are demarcated by boundary symbols "<l>" and "<w>" respectively. The CMU Pronouncing Dictionary, although extensive, does not include all vocabulary included in our dataset. To resolve this problem, *g2pE: A Simple Python Module for English Grapheme To Phoneme Conversion* (https://github.com/Kyubyong/g2p) was utilized which uses the CMU Pronouncing Dictionary and selects the correct pronunciation based in its POS tagging method. Pronunciations are then predicted for all out-of-vocabulary words based on learned probabilities of letter sequences.

Rhyme detection was then attempted using a one-to-one comparison of phonemes. Although many rhymes were detected, inaccuracies which cropped up in transcription complicated the task. Softer rhymes and the deliberate mispronunciation of words in rap (see Kanye West's *All Falls Down* when rhyming "secure" and "career") also add to the ambiguity of phonetic transcriptions simply based on the text of the lyrics. Although phonetic transcription was attempted on the acapella tracks of certain rap songs, this method was imprecise and computationally expensive.

One way to tackle this issue is to assign vector representation to each vowel phoneme in order to capture the similarity of pronunciation of two phonemes by calculating the distance between the two. (Figure 1) The vector representations were assigned by hand and the accuracy lacks validity. As we train our model, we found these representations can be learned in order to provide empirical similarities in a corpus of lyrics, shown in Figure 2. The original features of each phoneme are explained in the following section.

Figure 1: Vowels and its vectors

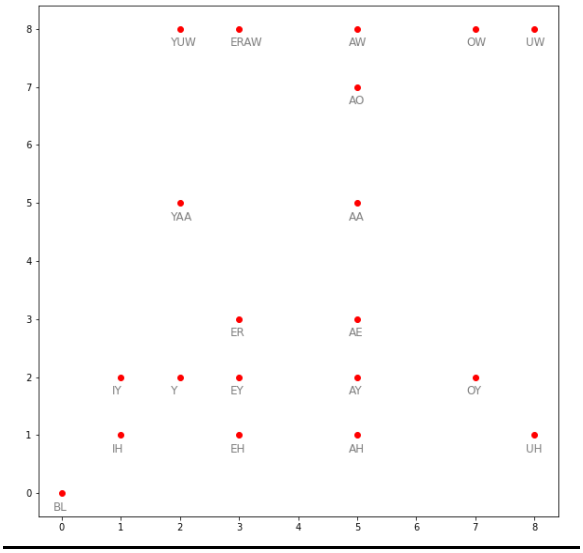
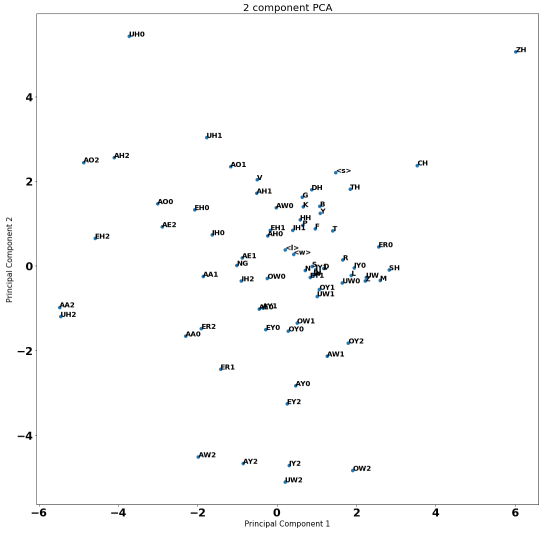


Figure 2: PCA of phonemes after training



While phonemic transcriptions alone provide a better understanding of the phonological qualities of words over a typical word-based language model, learning the poetic structure also requires attending to positions of phonemes and treating a phoneme differently if it is found at the beginning or end of a word than if it is found elsewhere.

4.2 Featurization

Analyzing the structure of rhymes gives us insight into the position of important phonemes. For example, in a part of Eminem's song, "Lose Yourself," we can see he lays out different types of rhymes, such as alliteration, assonance, and consonance, with a different level of density in just eight lines of lyrics.

Figure 3: Eminem's "Lose Yourself" Rhyming Structure

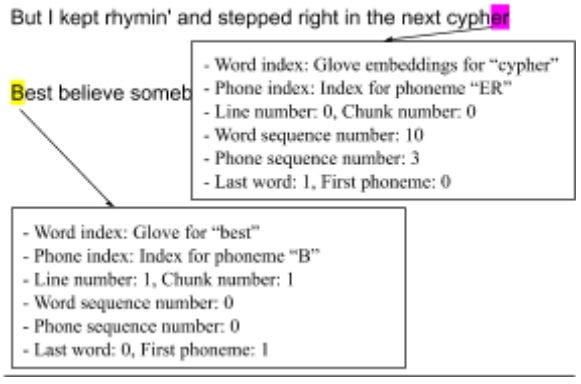
But I kept rhymin' and stepped right in the next cypher
Best believe somebody's payin' the Pied Piper
All the pain inside amplified by the
Fact that I can't get by with my nine-to-
Five and I can't provide the right type of life for my family
'Cause man, these goddamn food stamps don't buy diapers
And there's no movie, there's no Mekhi Phifer, this is my life
And these times are so hard, and it's gettin' even harder

We notice a number of distinct characteristics of rhymes in a rap song from the example above. First, rhymes work chunk-by-chunk. The first four lines end with the phoneme sequence "AI ER" and transition into irregular end rhymes in the next following four. Second, Each type of rhymes requires attending to different positions of phonemes. For example, in the set of alliterations shown in the second line, it is important that the "P" phoneme appears in the first position of each word within the proximate syllable window. (Payin the Pied Piper) Third, generally, the last words in each line carries a strong end rhyme (cypher, piper, by the, nine-to). Fourth, it is more likely that a set of internal rhymes is collocated in the same line. If two syllables are pronounced in the same way but placed in different lines, the probability that these two syllables rhyme is lower than when they are in the same line. Lastly, the positional information we observed above is dependent on being able to learn where the word and line boundaries are, which is derived by the structure of the text.

In addition to the sound devices, it is imperative for each phoneme to have semantic information similar to a word-based language model. This allows the model the ability to learn the rhyme structure and also understand the meaning of the word from which the phoneme derives. Features below are encoded to inform the rhyming structure and the semantics of words.

1) *Word embedding*: We use pre-trained, 100 dimension, GloVe word embeddings for vocabulary size of 500,000.
2) *Phone embedding*: A 100 dimensional vector for each of the 73 phonemes word/line/section boundaries. The vowel phonemes have stress indicators {0-2}.
3) *Line number*: Generally, a line represents a bar in a song (four beats). This can inform whether the phonemes are demonstrated in the same line.
4) *Chunk number*: We believe a chunk of four lines are a base-unit of rap lyrics. First line of the four is encoded as 0, the fourth line as 3, and the fifth line as 0, for example.
5) *Word sequence number*: N-th word of the line. It is useful in detecting the proximity of the phonemes.
6) *Phone sequence number:* N-th phoneme of the word. The model can learn where the phoneme is located in a word.
7) *Last word*: Whether the phoneme is a part of the last word of the line or not. It gives more attention to end rhymes.
8) *First phoneme*: Whether the phoneme is the first phoneme of the word or not. It gives more attention to alliterations.

Figure 4: Vector representation of phoneme (examples)



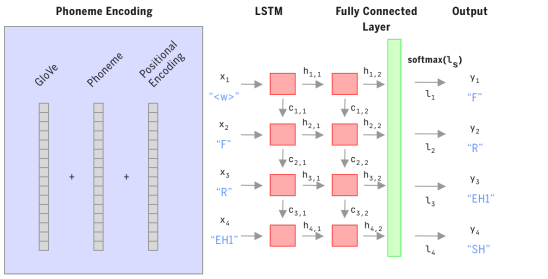
4.3 Model Structure

Architecture

Our model methodology is loosely based on the architecture found in *Character-Aware Neural Language Models* (Kim et al., 2015), employing two unidirectional LSTM layers with a hidden state size of 256. Our feature vector is of length 238 after concatenating the word embedding, phoneme embedding, and positional features. The phoneme embedding has a dimension size of 100, which was selected with cross validation. The embedding is initialized randomly and trained.

> **x**: *(sequence_length x batch_size x feature_size)* input vector representation at the phoneme level.
> **h**: *(2 x batch_size x 256)* hidden state at each time step for each LSTM layer.
> **c**: *(256 x batch_size)* cell state passed to subsequent timestep from LSTM layer.
> **l**: *(256 x 73)* fully connected layer.
> **y**: *(73 x 1)* predicted output for subsequent phoneme.

Figure 5: Model Diagram



Training

Over 25 epochs, using our dataset of over 31,000 songs, our model was trained on batches of phoneme of sequence length 250, roughly equating to 3 lines of lyrics. It achieved a cross-entropy loss of 0.68. By way of contrast, a more simplistic model trained on the same data, using only phonemic embeddings, achieved a cross-entropy loss of 1.72.

Decoding

Our output space for the model is much smaller than that of our intended use case. A sequence of ARPAbet tokens is not legible for the layperson. In order to generate lyrics, words must be transcribed from phonemes, back into words.

Sampling from the model is a stochastic process in which a distribution of the top *k* phonemes is sampled from at random. When a phoneme is selected, the output is fed back into the model for the next time step. During generation time, the word embedding is treated as an OOV token until a word boundary token is generated or an arbitrary *max_phone_length* is reached.

A greedy decoding algorithm takes a single vector output and uses it as input for the next timestep. This type of algorithm often results in sequences that don't translate directly to words. In order to improve this process, a modified fuzzy searching algorithm is employed. Given a sequence of length *m*, phoneme characters starting from the inside (*m/2*) are replaced by wildcard characters. This methodology preserves the characters important for alliteration and end rhymes, which occur at positions 0 and *m*.

While a greedy algorithm is computationally less demanding than other methods, there is significant information loss when a single phoneme is sampled. In order to retain this information, we employed a beam search algorithm. The algorithm starts when a non-boundary token is generated, and at each timestep, the output vectors are duplicated by a factor of the arbitrary *beam_width* parameter. Once a beam reaches a word_boundary character, its mean probability is calculated for each probability in the sequence and it is collected and removed from further calculations. Once all beams have terminated in a word boundary, or if an arbitrary *max_phoneme_len* is reached, all remaining beams are discarded and the collection of retained beams is used.

The remaining beams are then sorted by probability and the top sequence is selected. If the sequence can't transcribe directly to a word, then the subsequent sequence is selected until a word is generated. In the rare occurrence that a word is not produced, the top sequence is selected again and a similar fuzzy search algorithm is then employed as was the case in the greedy decoding algorithm.

Once a word is selected, the initial hidden states of the network are recalled and the correct input vectors with the newly acquired word embedding

are passed step-by-step. This process is repeated until a line boundary token is generated or an arbitrary *max_word_len* is reached.

5 Analysis

Given our outcome space is smaller than a word-based language model, our phoneme-based model trains much quicker with less computational demands. A comparable word-based model would also be limited by vocabulary size, resulting in less interesting results.

Because we make use of pretrained GloVe embeddings, we are able to fine tune these during training time, allowing for alternative representations for words inside the space of rap music.

In order to examine external validity of our model, we compare its word predictions to those of the simple baseline model we set up using Keras Sequential library. When given the same lines of lyrics, how different the predictions are and whether our model performs better in learning rhyming structure are the primary goal of the validation. The Keras model only uses sequences of lyrics as its input, with no existence of pre-trained embeddings or position information of phonemes and words. Figure 6 shows the Keras model's and our model's results, respectively.

Figure 6: Output lyrics

[Example1] Prime: Pots with cocaine residue
Every day I'm hustlin'
What else is a thug to do...
Keras Model: ...the day for my Westside freed I got a rhyme to die for my tonight and hi for the life s It s a next with a n***a in his
Our Model:...S IH1 S T AH0 Z <w> S T AH1 G N <w> T R UW1 S T <w> T AE1 K IH0 N <w> <l>
→ sistahz stand trust takin

[Example2] Prime: Okay, first things first, I'll eat your brains
Then I'ma start rocking gold teeth and fangs
'Cause that's what a motherfucking monster do
Keras Model: ... in the air dealers I be the the n***a of a n***a And out of my n***a to And And I m back to a gangsta So So I got
Our Model:...S T AE1 K S <w> B R IH1 T CH S <w> IH1 T S <w> M AA1 TH AH0 F AH2 K ER0 Z <w> <l>
→ stacks britches its muthafukers

While not statistically proven to be meaningful, our model shows that its understanding of the priming lyrics by repeating specific phonemes or a combination of phonemes that it observes in the priming lyrics. Compared to the Keras model, the outputs make more sense semantically and syntactically, although the number of observed tokens is limited. It is also shown that the model learns word boundaries and line boundaries defining both of the output length to be four-words long in these examples. We observed that the length is from four to five words long in most cases.

6 Conclusion

In this paper, we proposed a novel multi-layer encoder-decoder language model enabling attention to specific features in a rap song, namely, rhyming structure. We observed the outputs following the structure of our training and priming input data. Specifically, it creates a word that's similar to phonological characteristics of priming sequence. For example, the priming sentences have a dominant phoneme in it, the predictions are skewed towards the phoneme it sees in the priming lyrics. It also predicts word boundaries and line boundaries, and a certain type of structure that a set of phonemes needs in order to make it a word. (consonant - vowel - consonant, etc.) Moreover, its stochastic process allows more flexible structure compared to a word based language model, ability to provide creative solutions to exploring multi purpose language model and opens up the possibility for further development paired with a language model that pays attention to other dimensions of word generation, such as a semantic-focused language model.

This approach leaves room for improvement. First, the grapheme to phoneme transcription is based on common use of the language, undercutting various ways to pronounce, or mispronounce, a word for the sake of a rhyme, which is a distinct feature of rap compared to other literary genres. In future work, transcribing lyrics directly from audio data will allow us to work with more accurate pronunciation/ phonemes. Second, our model semantic coherence when generating lyrics. A major part of our work focuses on how to extract rhyming information, which leaves room for error at generation time, due to the inherent randomness. Lastly, the model can be more useful if it is trained on a bidirectional RNN model which enables more possibility for lyric generation.

This would allow the model to work for other applications, such as ‘fill in the blank’ or ‘word replacement’.

References

Eric Malmi, Pyry Takala, Hannu Toivonen, Tapani Raiko, and Aristides Gionis. 2015. *DopeLearning: A Computational Approach to Rap Lyrics Generation*. https://arxiv.org/pdf/1505.04771v1.pdf

Yoon Kim, Yacine Jernite, David Sontag, Alexander M. Rush. 2015. *Character-Aware Neural Language Models.* *https://arxiv.org/abs/1508.06615*

Justine Kao and Dan Jurafsky. 2012. *A Computational Analysis of Style, Affect, and Imagery in Contemporary Poetry.* *https://nlp.stanford.edu/pubs/kaojurafsky12.pdf*

Jack Hopkins and Douwe Kiela. 2017. *Automatically Generating Rhythmic Verse with Neural Network.* Proceedings of the 55th annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). 2017 https://research.fb.com/wp-content/uploads/2017/06/automatically-generating-rhythmic-6-2.pdf

Richong Zhang, Xinyu Liu, Xinwei Chen, Zhiyuan Hu, Zhaoqing Xu, and Yongyi Mao. 2019. *Generating Chinese Ci with Designated Metrical Structure.* AAAI2019. https://www.aaai.org/ojs/index.php/AAAI/article/view/4736

Nitish Shirish Keskar, Bryan McCann, Lav R. Varshney, Caiming Xiong, and Richard Socher. 2019. *CTRL: A Conditional Transformer Language Model for Controllable Generation.* https://arxiv.org/pdf/1909.05858.pdf

Hussein Hirjee and Daniel G. Brown. 2010. *Using Automated Rhyme Detection to Characterize Rhyming Style in Rap Music.* https://pdfs.semanticscholar.org/8b66/ea2b1fdc0d7df782545886930ddac0daa1de.pdf

Kento Watanabe, Yuichiroh Matsubayashi, Kentaro Inui, Tomoyasu Nakano, Satoru Fukayama, Masataka Goto. 2017. *LyriSys: An Interactive Support System for Writing Lyrics Based on Topic Transition.* https://staff.aist.go.jp/m.goto/PAPER/IUI2017watanabe.pdf

Peter Potash, Alexey Romanov, and Anna Rumshisky. 2015. *GhostWriter: Using an LSTM for Automatic Rap Lyric Generation.* https://www.aclweb.org/anthology/D15-1221

Folgert Karsdorp, Enrique Manjavacas, and Mike Kestemont. 2019. Keepin’ it real: Linguistic models of authenticity judgments for artificially generated rap lyrics. https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0224152

Hugo Gonçalo Oliveira. 2017. A Survey on Intelligent Poetry Generation:Languages, Features, Techniques, Reutilisation and Evaluation. https://www.aclweb.org/anthology/W17-3502.pdf