GROVER: Generating Rap by Observing VERses

Davis Haupt*, Arun Kirubarajan*, Jerry Lu*, Rohan Menezes*
University of Pennsylvania
{dhaupt, kiruba, lujerry, rohanmen}@seas.upenn.edu

Abstract

Recurrent Neural Networks are the current state of the art for language modelling tasks. However, it is difficult for such architectures to preserve certain linguistic characteristics of a given source text, namely rhyme patterns and meter (rhythm). In this paper, we examine several neural methods of generating rap lyrics by augmenting purely neural-based models by using external pronunciation and meter datasets. Our results show that these augmentations to standard Recurrent Neural Network succeed in learning modelling rhyme schemes as well as rhythm.

For reference, the code is available at https://github.com/davish/grover.

Our presentation can be found at http://tiny.cc/grover

1 Introduction

Hip Hop music has become the most popular genre of music in the United States. Lyrics are difficult to write, and artists often have a team of writers to help them produce their music. As such, there has been an increase in interest for generating new Hip Hop (hereby referred to as rap) lyrics. The task of automatically generating rap lyrics can be framed as a language modeling task, where a model can be trained on rap lyrics in order to generate additional lyrics. Certain linguistic difficulties arise from modeling rap lyrics. In particular, it is a non-trivial task to model rhyme schemes and meter (rhythm). This presents a challenge for non-rule based models to preserve such linguistic characteristics from the source text into the target text.

Due to an intrinsic similarity between rap lyrics and poetry, methods for generating poetry can be applied to solve the problem of maintaining rhyme and meter. The contribution of this paper is the demonstration of certain neural and linguistic mechanisms to improve the effectiveness of generating rap lyrics. In particular, we implement rhyme schemes using pronunciation heuristics and meter using an attention module in our LSTM architecture. In addition, we provide implementations of metrics for evaluating any kind of rhyme and meter-based text generation.

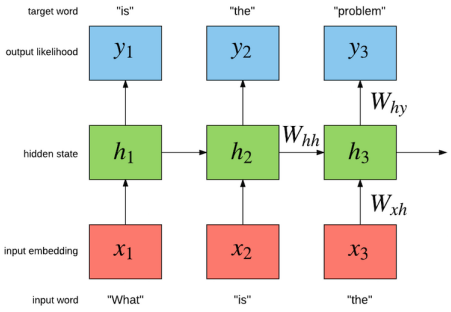

Figure 1: An RNN language model, which we use to produce the probabilities of a word appearing in a given context

2 Literature Review

In Unsupervised Discovery of Rhyme Schemes, Reddy and Knight (2011) aim to predict the rhyme schemes of poetry with no additional data than the text itself. While rhyming and pronunciation dictionaries can be useful for this task, they are extremely language (and region) specific. For example, the CMU pronunciation dictionary was made specifically for North American English. In order to label rhyme schemes onto their corpus, The authors describe a probabilistic model for generating rhyming stanzas. Working backwards, they then use the EM algorithm to learn rhyme schemes from the stanzas. Ultimately, they use their computed probabilities to label stanzas with

a rhyme scheme. Probabilities were computed for sub-corpora based on era, and compared to a standard rhyming dictionary. Interestingly, the model performed better than the rhyming dictionary on older poems before 1850, and worse than the rhyming dictionary on 20th and 21st century poetry. The authors explained this result through changing pronunciations in English over time. Rhyming dictionaries are specifically made for how English is pronounced today. Thus, those dictionaries would not be representative of pronunciations of English from more than 300 years ago. This shows that statistical learning of rhymes and rhyme schemes can be useful when looking at older texts such as Shakespeare, but that when trying to generate modern lyrics, it is a better decision to use existing rhyming dictionaries.

Potash et al. (2015) use an LSTM model end-to-end to generate rap lyrics. The goal of their system is to generate lyrics for a given artist that are similar enough to the source material that the similarities are clear, but different enough that the generated lyrics themselves do not appear in the original corpus. The authors of GhostWriter utilized a crucial observation from Karpathy et al. (2015), where an LSTM-based RNN generating C++ code was able to correctly close the curly braces of code, without being explicitly taught to. Building off this intuition allowed the authors to have an LSTM close each line with a rhyming word in a similar fashion by generating text using word embeddings. In a comparison to a Markov-based n-gram model, the authors found that the LSTM was able to produce rap lyrics with a high TF-IDF similarity to the source material, while maintaining a higher rhyme density. This system serves to show that purely neural methods are sufficient to create rhymes within generated test.

Instead of only using an RNN, Ghazvininejad et al. (2016) developed Hafez, a poetry generation system which uses a combination of hard constraints and a RNN Language Model (LM) to generate realistic poetry. In order to produce poetry with good rhyming and meter, Hafez uses a Finite State Acceptor (FSA) which is built to accept only stanzas which fit the desired meter and the desired rhyme scheme, which for the purpose of the paper is that of a Shakespearean sonnet with iambic pentameter. First, a topic is input manually, and a vocabulary of words is chosen via word2vec similarity. These words are put into rhyme groups, and a rhyme scheme is then chosen. Working backwards from the end of each line, the rest of the line is generated via Beam search, using the probability of the word given from the RNN along with whether or not it is a valid transition in the FSA. Through this method, convincing English poems can be generated with good rhyme schemes that scan well.

3 Experimental Design

3.1 Data

We had three data sources. The first was the set of all of Kanye West's lyrics, scraped from azlyrics.com. The second was a large dataset of hip-hop lyrics extracted from a larger Kaggle dataset of 380,000 song lyrics. These two datasets had different benefits and drawbacks. The Kanye corpus provided us with a much more focused and consistently formatted dataset on which to train our models. The Kaggle corpus, conversely, had a much larger amount of data, granted in a slightly less consistent format. Additionally, the combination of many artists' styles into one language model may have had effects which should be investigated further on.

While developing our model, we found that the lyrics did not make much sense, nor did they reflect typical English syntax. Thus, we wanted to train our model on a larger corpus of typical English text, then transfer that language model onto the corpora of rap lyrics. For this task, we used the English Gigaword corpus (Graff and Cieri, 2003).

Corpus	Size	Contents
Kanye West	0.5 MB	Only Kanye West (from azlyrics.com)
Kaggle Lyrics	53 MB	Lyrics by various hip-hop artists
Gigaword	12 GB	English news-wire

Table 1: Description of the data used to train our language models

3.2 Evaluation Metric

In addition to subjective evaluation of the verses produced, we used perplexity as a method of in-

trinsic evaluation. We used the formula for perplexity described in Homework 5 and by Chen et al. (1998):

$$Perplexity(w) = P(w_1 w_2 \dots w_n)^{-1/N}$$
$$= \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 w_2 \dots w_{i-1})}}$$

To perform this calculations, we used the word probabilities from our n-gram language model described below in §3.3 below.

The evaluation script takes two inputs: A corpus of rap lyrics, and a string generated by our model. The script outputs the perplexity of that string, and a string with a lower perplexity is more similar to the lyrics in the corpus than a string with a higher perplexity.

3.3 Simple Baseline

Our sample baseline is a simple n-gram language model from the `markovify` python package. Here is some example lines from that model, trained on all of our Kaggle hip-hop data:

```
Ja ja ja!
<NEWLINE>
Hate is the Slavic blood, this is
not always showing Am I seeing
truth, or do I look down in the
room Maybe death and the boys
in the studio pranksters Yo
lookin at me, sideways,
sideways, sideways Them haters
lookin at me, but I'm set straight
like my style?
<NEWLINE>
Go and bust tonight?
<NEWLINE>
Yo we back on the 12th of
September You better off with,
I love them, just to kick it to
the West coast, nigga can come
around Bossman!
<NEWLINE>
Shawntae nigga Would you
cook for me?
```

Using the model itself as a base, this sample had a 1.8 perplexity score. One issue we immediately see in this stanza is that lines can grow to be very long. Line 2 in particular is much longer than any line found in typical rap lyrics. This deficiency in a purely generational approach to lyric generation is tackled in §4.2.2.

4 Experimental Results

4.1 Published baseline

The published baseline we chose to compare our model to is a Recurrent Neural Network that only uses the content of rap lyrics in order to generate rhymes. The specific implementation who chose to compare to is the LSTM model used by Potash et al. (2015). We used an implementation of this architecture in PyTorch trained on the Kanye West lyric set to compare our model to.

4.2 Extensions

4.2.1 Attention

Attention is a popular and effective technique in NLP nowadays and we wanted to see whether it would help for our rap generation. Specifically, we tried out input attention. In a typical model, when generating the next word of our rap, we would simply feed in the embedding of the previous generated word. However, in the new attention model, we would individually feed each already generated word of the rap (along with the current hidden state) into an attention module. The module was simply a linear layer that transformed the concatenated combination of a single generated word and the hidden state into a single real number. We then took a softmax over all of the returned numbers to compute weights for each already generated word. We then determined our input to the next time step of the RNN by multiplying each already generated word embedding by its weight and then summing all the word embeddings together.

However, attention unfortunately did not help us in our generation. The positive results from attention have previously come from sequence to sequence models, where you could attend over the entire input sequence. However, in this particular case of generation, there is no possible way to know future words, so there was no way to attend over them. Thus, the number of words we were attending over varied widely (extremely few for the first few words and a lot for the last few words). This likely made it pretty hard for the attention model to learn appropriate weights for each word based on this variable length. Also, when youre generating a next word of a sequence, its more important for that word to make sense in relation to the previous word (which is what a no attention model would encourage) than

to make sense in conjunction with long range dependencies (which is what an attention model would encourage). Essentially, the local grammar of a sentence is more important to making a good rap than reincorporating words from farther away. And finally, incorporating other words makes the ultimate input to the RNN time step extremely similar to the inputs of the immediately preceding time steps (since they are all weighted averages of largely the same vectors). Since the inputs are so similar, the RNN time step would tend to output the same word for all the inputs. Thus, the RNN with attention tends to repeat words for about three time steps in a row.

4.2.2 Rhyme and Meter

Song lyrics typically have a pattern of rhyming and rhythm which make the lyrics more interesting to listen to. We wanted to augment our RNN such that it would output rap lyrics which have end rhymes and follow an established meter. To enforce a consistent meter, we selected a meter to be used line-by-line (e.g. Iambic or Trochaic Pentameter). This meter would be encoded as a series of 1s and 0s, where 1 represents a stressed syllable and 0 represents an unstressed syllable. For example, we produced several lyrics with trochaic pentameter, which would be encoded as `1010101010`. To find the stress pattern of a given word, we used the CMU pronunciation dictionary. This dictionary gives the precise phones in the pronunciation of a given word, which we use to find rhyming words, as well as the locations of primary and secondary stress. For our purposes, we only care whether a syllable is stressed, so we treated primary and secondary stress as the same.

Given this information, we track of the sequence of stressed/unstressed syllables expressed by the line so far. Then use the RNN to generate a word based on the context. If the stress pattern of this words fits into our target meter, then accept it as the next word in the line, otherwise, choose a new word probabilistically. When no more syllables are left on the current line, insert a new line and reset the target sequence of syllables. We also enforce that our model produces rhyming couplets. We do this by setting a target rhyme, which is the empty string or the last word in the previous line. If a word is the last word in the line, we alternate between enforcing that it rhymes with the previous line, or accepting it regardless of rhyme, and the next line will rhyme with the current word. As with stress information, whether a word rhymes with another word can be determined by querying the CMU Pronouncing Dictionary.

4.3 Model Training

We trained our RNN on two types of data. We first primed the RNN by training it on a subset of the English Gigaword Corpus for ten thousand epochs. After priming, we then trained the model for one thosand epochs on Kanye West's lyrics, with a doubled learning rate. The hope here was that the model could learn first how English is structured grammatically, before learning the specifics of rap lyrics.

We thought this to be an important step, considering how rap lyrics can be different from normal speech. Drawing on "Heavy Metal and NLP" (Barr, 2016), we define a word w's "Kanyeness" to be the log-ratio of frequency in the Kanye corpus vs the standard Gigaword corpus:

$$K_w = \log\left(\frac{N_w^{kanye}}{N_w^{gigaword}}\right)$$

Of course, this metric is only defined when words appear in both corpora. Additionally, words must appear at least five times in each to have a well-defined Kanye score.

Rank	Most	Least
1	sweat	last
2	sting	also
3	coupons	US
4	Never	I
5	hazard	government
6	neighbourhood	two
7	Chrysler	year
8	script	would
9	jam	The
10	vocal	said

Table 2: The ten most and least "Kanye" words

4.4 Results

Below is an example poem generated by GROVER:

Plenty pimps ballistics Aussie Wickert
Medecin ballistics Yacht enticing
purge obeyed approaches pimps ballistics
nonsense Medlin Hendrik Medlin booed statistics
alley alley beams arisen catch catch
Desmond Desmond kills estates estates batch
Desmond fearing undefined ballistics
vital shore existing Desmond logistics
Aussie D. enticing Aussie vital
Wickert Desmond catch concession Cox recital

4.5 Error analysis

Analyze error in best performing system, show examples of the errors. Can you categorize the types of errors that it makes, and give an estimate of how prevalent each error type is? If you extensions performed better than the published baseline, then show examples of the errors that the published baseline makes that your extensions get correct (and vice versa if your extension introduces some new errors).

The words in the lyrics our model generates scan well and rhyme convincingly, but seem to be chosen at random. Furthermore, the probability distribution among the vocabulary seems to be fairly constant, regardless of the hidden state. When we trained our model on lyrics from a narrow subset of artists, the limited vocabulary causes certain words not to have proper rhymes. Thus, we encountered cases where our model output lines that do not rhyme, for example:

```
the found forgot reminds
dismiss umbrella <newline>
oath opponent kissing
stole celestial <newline>
```

As discussed in §4.2.1, attention on its own was not found to help. The number of words we were attending over varied widely, and for our task, it was important for words to make sense in relation to the previous word. Incorporating other words makes the ultimate input to the RNN time step extremely similar to the inputs of preceding time steps. Thus, the RNN tended to output the same word for all inputs.

5 Conclusion

In this paper, we present a generalizable approach to generating text that follows some rhyme scheme and meter pattern. Some future work would include applying the methods we implemented to the domain of lyric generation for other genres of music (e.g. country, pop). In addition, this model architecture could be extended to a Seq2Seq network in order to create a dialogue system with the capabilities of conversing in lyrics.

Acknowledgments

Kanye West.

References

Iain Barr. 2016. Heavy metal and natural language processing.

Stanley F Chen, Douglas Beeferman, and Roni Rosenfeld. 1998. Evaluation metrics for language models.

Marjan Ghazvininejad, Xing Shi, Yejin Choi, and Kevin Knight. 2016. Generating topical poetry. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1183–1191.

David Graff and Christopher Cieri. 2003. English gigaword. *Linguistic Data Consortium, University of Pennsylvania*.

Peter Potash, Alexey Romanov, and Anna Rumshisky. 2015. Ghostwriter: Using an lstm for automatic rap lyric generation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1919–1924.

Sravana Reddy and Kevin Knight. 2011. Unsupervised discovery of rhyme schemes. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 77–82.

A Appendices

Appendices are material that can be read, and include lemmas, formulas, proofs, and tables that are not critical to the reading and understanding of the paper.