GhostWriter: Using an LSTM for Automatic Rap Lyric Generation

Peter Potash, Alexey Romanov, Anna Rumshisky
Dept. of Computer Science
University of Massachusetts Lowell
Lowell, MA 01854
{ppotash,aromanov,arum}@cs.uml.edu

Abstract

This paper demonstrates the effectiveness of a Long Short-Term Memory language model in our initial efforts to generate unconstrained rap lyrics. The goal of this model is to generate lyrics that are similar in style to that of a given rapper, but not identical to existing lyrics: this is the task of ghostwriting. Unlike previous work, which defines explicit templates for lyric generation, our model defines its own rhyme scheme, line length, and verse length. Our experiments show that a Long Short-Term Memory language model produces better "ghostwritten" lyrics than a baseline model.

1 Introduction

Ghostwriting defines a distinction between the performer/presenter of text, lyrics, etc, and the creator of text/lyrics. The goal is to present something in a style that is believable enough to be credited to the performer. In the domain of rap specifically, rappers sometimes function as ghostwriters early on before embarking on their own public careers, and there are even businesses that provide written lyrics as a service [1]. The goal of GhostWriter is to produce a system that can take a given artist's lyrics and generate *similar* yet *unique* lyrics. To accomplish this, we must create a language model to produce text, while also understanding what 'style' means in a quantitative sense.

The contribution of this paper is three-fold: (1) we present the ghostwriting problem of producing similar yet different lyrics; (2) we present computational, quantitative evaluation methods for these two aspects; (3) we evaluate the performance of a Long Short-Term Memory (LSTM) vs n-gram model for this problem.

2 Related Work

Recent work (Sutskever et al., 2011; Graves, 2013) has shown the effectiveness of Recurrent Neural Networks (RNNs) for text generation. In their works, the authors use an RNN to create a language model at the character level. The results are inspiring, as the models learn various grammatical and punctuation rules, such as opening and closing parentheses, plus learning a large vocabulary of English words at the character level. Graves (2013) uses a variation of an RNN called LSTM architecture which creates a better language model than a regular RNN.

Text generation for artistic purposes, such as poetry and lyrics, has also been explored, often using templates and constraints (Oliveira et al., 2014; Barbieri et al., 2012). In regards to rap lyrics, Wu et al. (2013) present a system for rap lyric generation that produces a single line of lyrics that are meant to be a response to a single line of input. However, the work that is most similar to ours is that of Malmi et al. (2015). The authors create fixed 16-line verses, generating the verse line-by-line using full lines from existing rap songs. The system predicts the best next line based on the previous lines, using a system that records an 81.9% accuracy predicting next lines in already existing verses. The feature that provides the greatest accuracy gain is a neural embedding of the lines, created from the character level.

Hirjee and Brown (2010b) have developed a rhyme detection tool based on a probabilistic model (Hirjee and Brown, 2010a) that analyzes phoneme patterns in words. The model is trained on a set of lyrics that were manually annotated for rhyming words. The statistics generated by the rhyme detection tool will be an important part of

[1] http://www.rap-rebirth.com/, http://www.precisionwrittens.com/rap-ghostwriters-for-hire/

Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 1919–1924,
Lisbon, Portugal, 17-21 September 2015.

our evaluation (see Section 5).

3 Generating Lyrics

In a departure from previous work on poetry/lyric generation, our goal is to build a model that does not require templates/constraints to generate lyrics, while also being able to produce full verses, as opposed to single lines. The system must be able to model general human language in order to produce fluent lyrics, but it must also be able to model the style of a target artist, by understanding the artist's vocabulary and rhythmic style, in order to fully execute the ghostwriting task of producing similar yet new lyrics.

3.1 LSTM

Here we will give a very brief overview of RNNs and LSTMs. For a more detailed explanation please refer to (Graves, 2013). The foundation of an RNN (of which an LSTM is specific architecture) is a word embedding E that provides a vector representation for each of the words in our corpus. Given a history of words $w_k, ..., w_0$ we want to determine $P(w_{k+1}|w_k, ..., w_0; E, \Phi)$, where Φ is a set of parameters used by our model. In the context of an RNN we define this probability by:

$$P(w_{k+1}|w_k, ..., w_0; E, \Phi) = f(x, s) \quad (1)$$

At each time-step the RNN computes f given an observation x and a previous state s. The input goes through a transformation where it passes through one or several hidden layers.

The LSTM model uses a specific architecture for the hidden transformation, defined by the LSTM memory cell. The key feature to the LSTM memory cell is the presence of an input gate, output gate, forget gate, and cell/cell memory, which manifest themselves in the model as activation vectors. Each of these gates/cells has its own bias vector, and the hidden layer at each time-step is now a complex nonlinear combination of gate, cell, and hidden vectors.

3.2 Using LSTM for Lyrics Generation

Since previous work has shown the power of RNNs to model language, we hope that it can capture the rhythmic style of an artist by learning rhyme and meter patterns. As noted in Section 2, LSTMs have performed well at sequence forecasting, for example at learning punctuation, such as opening and closing parentheses. We see the task of rhyme detection as something similar in nature. Kaparthy et al. (2015) have also shown that LSTMs could successfully learn where to place the brackets and indentation in C++ code. In their model, certain LSTM cells activated specifically when encountering end of the line. We believe learning rhymes at the end of the line is conceptually similar to such tasks.

3.3 Verse Structure and Rhyme Inference

The goal of our model is to not just generate lyrics, but generate the structure for the lyrics as well. To do this, we have added "<endLine>" and "<endVerse>" tokens to the lyrics. From this, the system will generate its own line breaks, while also defining when a generated verse ends. This allows us to analyze non-rhyming features from (Hirjee and Brown, 2010a), such as number of syllables per line and number of lines per verse. We also desire that, by using the "<endLine>" token, the system has a better chance of understanding rhyme schemes used by an artist. For example, the LSTM can capture the pattern of "came <endLine>" followed shortly by "name <endLine>" to understand that "came" and "name" are a rhyming pair. To do this effectively, the system would need sufficient training data where rhyming pairs occur frequently enough to actually dictate a pattern, similar to (Reddy and Knight, 2011; Addanki and Wu, 2013).

4 Experimental Design

4.1 Dataset

We collected songs from 14 artists from the site *The Original Hip-Hop (Rap) Lyrics Archive - OHHLA.com - Hip-Hop Since 1992*[2]. In the present lyrics generation experiments, we used the lyrics from the rapper Fabolous. For training, we used 219 verses with at least 175 words in each verse. We selected Fabolous because his lyrics produced the highest accuracy in the artist recognition experiments in (Hirjee and Brown, 2010a). We conjecture that because of this, he had the most consistent style, making him a good choice for initial experiments.

[2] http://www.ohhla.com/

4.2 Baseline

To compare with the results of the LSTM model, we followed the work of (Barbieri et al., 2012) and created a Markov model for lyric generation. Since the goal of our work is to make an unsupervised system, we do not use any constraints or templates to produce the lyrics. Thus, our baseline simplifies to a basic n-gram model. Given a history of w_{k+n-1},...,w_k, the system generates a new token t as follows:

$$P(w_{k+n} = t|w_{k+n-1}, ..., w_k) = \frac{|w_k,...,w_{k+n-1},t|}{|w_k,...,w_{k+n-1},\bullet|} \quad (2)$$

where $|w_k...w_{k+n-1}t|$ is the amount of times the the context w_{k+n-1},...,w_1 is followed by t in the training data, and $|w_k...w_{k+n-1}\bullet|$ is the amount of times the context appears followed by any token. There is the possibility that the context has never been encountered in the training data. When this occurs, we back off to a smaller n-gram model:

$$P(w_{k+n} = t|w_{k+n-2}, ..., w_k) = \frac{|w_k,...,w_{k+n-2},\bullet,t|}{|w_k,...,w_{k+n-2},\bullet,\bullet|} \quad (3)$$

The model may have to back-off multiple times before it encounters context it has seen in the training data. Once we back-off to the point where we compute $P(w_{n+k} = t|w_k)$, we are guaranteed to have at least one non-zero probability, because w_k must have appeared in the vocabulary for it to have been generated previously.

Note that rather than backing off to a lower-order n-gram model, we use a skip-gram model which drops the words immediately preceding the predicted word. The main motivation for this is that it allows us to capture long-range dependencies, which makes it into a better baseline comparison for an LSTM.

4.3 Model Initialization

When producing lyrics with either the LSTM or baseline model, we initialize with the "<startVerse>" token. Once the model produces a token, it becomes part of the context for the next step of token generation. Our models are closed in the sense that they only produce tokens that appear in the training vocabulary.

4.4 LSTM Implementation

We used a Python implementation of an LSTM from Jonathan Raiman[3]. The LSTM is built on top of Theano (Bastien et al., 2012; Bergstra et al., 2010). Following (Graves, 2013), we set the amount of LSTM inputs/outputs to be equal to the vocabulary size. Also, to avoid the vanishing gradient problem when training RNNs, we clip the gradients in the range [-1,1]. We train our LSTM model using a Tesla K40 GPU on a single workstation.

5 Evaluation Methods

In this section, we present automated methods for evaluating the quality of generated lyrics. Ideally, judging system output in terms of, e.g. fluency, should be conducted using manual evaluation. However, conducting formal human evaluation is somewhat problematic. For a full *qualitative* evaluation of a given artist that would assess both similarity of style and novelty, the evaluator would need to know that particular artist's body of work very well. Even finding annotators who are well-versed in the general art of rap lyrics can be challenging (Addanki and Wu, 2014). While this may be possible for the present experiments that focus on a single artist, it is hardly feasible for larger-scale studies that will use our full data set that contains the lyrics of 14 different artists. We therefore propose an automated evaluation method which we believe is able to capture two critical aspects of ghostwriting, which are in fact quite tricky to capture together: being similar, yet different.

5.1 Similarity to existing lyrics

In order to evaluate the novelty of generated lyrics, we compare the similarity of the generated lyrics to the lyrics in our training set. We used an algorithm proposed by (Mahedero et al., 2005) for calculating the similarity between produced lyrics and all verses from the same artist. This algorithm is based on the well-known Inverse Document Frequency, using cosine on document vectors to calculate distance. First, we build the Term-Document Matrix with weights for each term in each song:

$$w_{ij} = f_{ij} log(\frac{N}{n_j}) \quad (4)$$

[3] https://github.com/JonathanRaiman/theano_lstm

where N is the total number of documents (verses, in our case), n_j is the number of verses that contains term j and f_{ij} is the frequency of term j in the ith verse. Using this matrix, we can calculate the cosine distance between verses and use it as a measure of similarity. When discussing similarity, we refer to the max similarity: of all verses it is most similar to, exactly *how* similar is it? The lower the max similarity score, the more novel the lyrics.

5.2 Numerical features of the lyrics

We also produced the features from (Hirjee and Brown, 2010a) for our generated verses. The statistics of these features are meant to represent how effective we are in modeling an artist's style. The point of the system is not to produce arbitrary rhymes; it is to produce rhyme types and rhyme frequency that are similar to the target artist. Following (Malmi et al., 2015), the rhyme feature we examine in this work is rhyme density. Rhyme density is defined as the total number of rhymed syllables divided by the total number of syllables.

6 Results

For the lyrics generation experiments reported here, we used the rapper Fabolous as the artist whose style we tried to emulate.

6.1 Example of Generated Lyrics (Warning: Explicit Content)

Below is a sample of lyrics generated by the LSTM model:

> Line 1: i m old enough to know better young enough to not give a fuck
> Line 2: rather hold my head high and die then live and duck
> Line 3: i got ta fuck be up and little niggaz go in
> Line 4: when i m in the feds and scoped by uncle sam
> Line 5: dope and hunn ed grams rope and hunn ed grams
> Line 6: at the same time she jerking and wiggling
> Line 7: smirking and giggling

While the pairs of rhyming end-lines in the generated lyrics are taken from the training data (the max similarity is 0.41), no more than two lines appear from a single verse. Though the generated lyrics aren't novel in a strict sense, the LSTM model is more effective than the n-gram model at using lines from multiple verses (see next Section 6.3). The rhyme density of this verse is 0.35, which is almost equal to Fabolous's average rhyme density (0.34).

6.2 Quantitative Analysis

As previously mentioned, the key to effective ghostwriting is to mirror an artist's style, but also providing original content. While vocabulary and lyrical content are key components for an artist's style, this is inherently satisfied by using words only from the training data. Thus, rhyme style – specifically rhyme density – will be the key performance indicator for imitating an artist's style. In terms of rap lyrics in general, a higher rhyme density is often better. Therefore for our system we would like a high rhyme density, but with a low max similarity score (a higher novelty). Figures 2 and 1 show the graph for rhyme density and max similarity for the LSTM and n-grams models, respectively. For the LSTM model the values are graphed compared to training iteration number – as the model becomes more fit to the data. For the n-gram model they are graphed dependent on n-gram value. For each n-gram value, we generate 10 verses and compute the average value of the two metrics. One expects that a perfectly fit LSTM model without regularization would exactly reproduce lyrics from the training data, and a high n-gram value would would also produce duplicate lyrics. This is evident in the graphs.

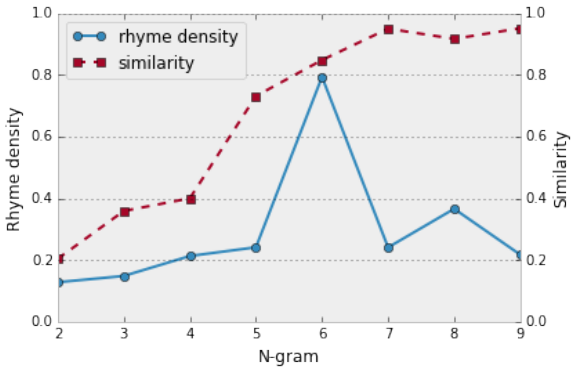

Figure 1: Values of rhyme density and max similarity versus n-gram value for the n-gram model.

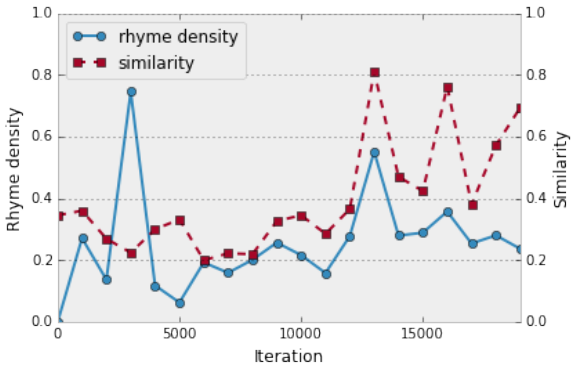

Figure 2: Values of rhyme density and max similarity versus iteration number when training the LSTM model.

6.3 Correlation of Rhyme Density and Max Similarity

Since exact replication would assuredly give a higher rhyme density than randomly produced lyrics, we desire a *low* correlation between rhyme density and max similarity. The correlation between rhyme density and max similarity for the LSTM model is 0.32, and for the n-gram model it is 0.47. When examining Figures 1 and 2 one may notice the anomalous points of high rhyme density (at $n = 6$ on the n-gram graph and 3,000 iterations for the LSTM model). After further inspection of the lyrics at these points, we see the lyrics contain repetitions of the exact same phrase. Since words are repeated frequently, the rhyme density of the lyrics is high (repeated words create rhymed phonemes, according to the rhyme detection tool). These points cause the similarity-density correlations to be artificially lower. After removing these data points, the LSTM model still has a lower correlation than the n-gram model, but the gap is much smaller: 0.71 compared to 0.75. Ultimately however, this shows that the LSTM model is better at generating original, rhyming lyrics.

6.4 Style Matching

Unfortunately, the correlation numbers do not dictate specifically the effectiveness of the LSTM model in the ghostwriting task. Instead, we can look at that max similarity values of both systems when they generate lyrics that produce a rhyme density similar to the average rhyme density of the target rapper. Looking at 100 randomly selected verses, Fabolous has an average rhyme density of 0.34. To do our analysis, first we create four regression lines, one for each metric (max similarity and rhyme density) in each model (we do not include the points of high rhyme density). Next we use the two rhyme density lines to determine at which iteration/n value the systems generate a rhyme density of 0.34. After that we plug these numbers into the two similarity lines to determine what similarity is needed to achieve the target rhyme density. The n-gram model line has a similarity of 1.28 at this point (above the max value of 1 for the metric), while the LSTM model has a value of 0.59. Based on these numbers, the LSTM model clearly outperforms the n-gram model when it comes to making original lyrics that are similar in style to our target rapper.

7 Conclusion

In this work, we have shown the effectiveness of an LSTM model for generating novel lyrics that are similar in style to a target artist. We compare the performance of the LSTM model to a much simpler system: an n-gram model. The results of our experiments show that, as an unsupervised, non-template model, the LSTM model is better able to produce novel lyrics that also reflect the rhyming style of the target artist. In future work, we plan to use more data to train our model, making it easier for our system to actually identify rhyming pairs and use them in new contexts. We also plan to encode phoneme features of words to improve rhyme discovery. Furthermore, we plan to generate lyrics from artists with a varying vocabulary size to see if it is easier to generate lyrics for an artist with a smaller vocabulary. In terms of evaluation, we hope to incorporate some method to evaluate the fluency of generated lyrics (Addanki and Wu, 2014). Lastly, to further avoid over-fitting to the training data and reproducing lyrics with a high similarity, we plan to use weight noise (Jim et al., 1996) to regularize our model.

Acknowledgments

We would like to thank the anonymous reviewers for their feedback.

References

Karteek Addanki and Dekai Wu. 2013. Unsupervised rhyme scheme identification in hip hop lyrics using hidden markov models. In *Statistical Language and Speech Processing*, pages 39–50. Springer.

Karteek Addanki and Dekai Wu. 2014. Evaluating improvised hip hop lyrics–challenges and observa-

tions. In *Proceedings of The Ninth International Conference on Language Resources and Evaluation (LREC)*.

Gabriele Barbieri, François Pachet, Pierre Roy, and Mirko Degli Esposti. 2012. Markov constraints for generating lyrics with style. In *ECAI*, pages 115–120.

Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian Goodfellow, Arnaud Bergeron, Nicolas Bouchard, David Warde-Farley, and Yoshua Bengio. 2012. Theano: new features and speed improvements. *arXiv preprint arXiv:1211.5590*.

James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. 2010. Theano: a cpu and gpu math expression compiler. In *Proceedings of the Python for scientific computing conference (SciPy)*, volume 4, page 3. Austin, TX.

Alex Graves. 2013. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*.

Hussein Hirjee and Daniel Brown. 2010a. Using automated rhyme detection to characterize rhyming style in rap music.

Hussein Hirjee and Daniel G Brown. 2010b. Rhyme analyzer: An analysis tool for rap lyrics. In *Proceedings of the 11th International Society for Music Information Retrieval Conference*. Citeseer.

Kam-Chuen Jim, Clyde Lee Giles, and Bill G Horne. 1996. An analysis of noise in recurrent neural networks: convergence and generalization. *Neural Networks, IEEE Transactions on*, 7(6):1424–1438.

Andrej Karpathy, Justin Johnson, and Fei-Fei Li. 2015. Visualizing and understanding recurrent networks. *CoRR*, abs/1506.02078.

Jose PG Mahedero, Álvaro MartÍnez, Pedro Cano, Markus Koppenberger, and Fabien Gouyon. 2005. Natural language processing of lyrics. In *Proceedings of the 13th annual ACM international conference on Multimedia*, pages 475–478. ACM.

Eric Malmi, Pyry Takala, Hannu Toivonen, Tapani Raiko, and Aristides Gionis. 2015. Dopelearning: A computational approach to rap lyrics generation. *arXiv preprint arXiv:1505.04771*.

Hugo Gonçalo Oliveira, Raquel Hervás, Alberto Díaz, and Pablo Gervás. 2014. Adapting a generic platform for poetry generation to produce spanish poems. In *5th International Conference on Computational Creativity, ICCC*.

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: short papers-Volume 2*, pages 77–82. Association for Computational Linguistics.

Ilya Sutskever, James Martens, and Geoffrey E Hinton. 2011. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024.

Dekai Wu, Karteek Addanki, Markus Saers, and Meriem Beloucif. 2013. Learning to freestyle: Hip hop challenge-response induction via transduction rule segmentation. In *2013 Conference on Empirical Methods in Natural Language Processing (EMNLP 2013), Seattle, Washington, USA*.