Re-implementing Potash et al.'s GhostWriter: Using an LSTM for Automatic Rap Lyric Generation

Grace Kim, Tara Kazemi
Dept. of Computing Science
Simon Fraser University
Burnaby, BC V5A 1S6
{tka51, thassanz}@sfu.ca

Abstract

This project explores the effectiveness of a Long Short-Term memory language model in an effort to generate rap lyrics. The goal of this model is to generate lyrics that are similar in style to a particular rapper but not identical to existing lyrics. We compare the LSTM generated lyrics to that of a simple RNN model. Our experiment will show that a Long Short-Term memory language model can produce ghost-written lyrics better than a baseline model.

1 Introduction

This project aims to reimplement a paper by Potash et al. (2015), which discusses previous research that performs lyric generation using Recurrent Neural Networks (RNN) and builds on that research by exploring the extent to which Long Short Term Memory networks (LSTM) can generate song lyrics and evaluates the results in comparison to the training data set.

We decided to focus our project on a Ghostwriting algorithm because not only were we intrigued by the concept of lyric generation, but we also believe it shows great promise for artists who are experiencing "writer's block" and wish to use the services of a ghostwriting program to spark their creativity.

Our models were trained using lyrics from various artists, take the pre-processed lyrics as input and generate a new set of verses that are unique yet similar to the artist's style.

2 Related Work

Lyrical generation with different RNN models have been explored in the past. Among them, Potash et al. (2015) was the first to explore the idea of implementing the LSTM model to generate lyrics, and evaluate the effectiveness of the model using quantitative evaluation metrics. This model was further enhanced by Potash et al. (2018) later on with new evaluation methodology. Similarly, Malmi et al. (2016) explored the capabilities of finding relevant next line from the existing data set using information retrieval techniques to create a new set of lyrics.

However, the models developed by Potash et al. (2015) and Malmi et al. (2016) were limited in that they were unconditioned and the lyrics were synthesized without context. Nikolov et al. (2020) further enhanced this model by implementing a conditioned model that trains a Transformer-based auto encoder to reconstruct rap lyrics, so that any text can be used for lyric generation later on. In other words, for conditioned models, the lyric generator is provided with a source text of any context and tasked with transferring the style of the text into rap lyrics.

3 Approach

This project aims to show that an LSTM model with a word embedding layer can generate unique lyrics that resemble the artist's style than a baseline model. Since previous research (Potash et al., 2015) shows that the LSTM models are much more efficient than n-gram models in generating lyrics, we implemented the simple RNN as the baseline model. Then, we explored the capabilities of LSTM and the advanced LSTM model.

The approach closely follows the steps outlined by Potash et al. (2018), and the tutorial by Wezley Sherman[1] which was inspired by his paper. However, rather than directly copying the code, we took the ideas from his notebook, which used Keras[2], and re-implemented them in PyTorch[3].

[1] https://github.com/wezleysherman/RNN-Rap-Lyric-Generator
[2] https://keras.io/
[3] https://pytorch.org/

There were three main steps in our approach. 1) Data Processing, 2) Training the model and 3) Generating Lyrics. Each of these steps are explained more in detail in the following sections. Our models were trained using a Tesla K80 GPU on Google Colab[4].

3.1 Simple RNN

Recurrent Neural Networks are a type of neural network used for analysing sequences because their architecture allows them to take previous outputs as inputs as well as the hidden state from the previous word and produce the next hidden state, for each word in the sequence. (Sherstinsky, 2020)

The Simple RNN has a wide applicability in NLP programs, since it work well with sequence data. However, the Simple RNN does have its limitations, over time, as the model is training, it will encounter the Vanishing Gradient Problem. This means the network will not be able to update all of its weights, some earlier weights will not be expressed, this impacts further weights throughout the network.

The model's function is expressed mathematically as:

$h_{(t)} = tanh(b_{hidden} + W_{hidden}h_{(t-1)} + W_{input}x_{(t)}$

Our model has three layers, an embedding layer, a linear layer, and our RNN. The embedding layer is added to allow for inputs to be transformed from the one-hot vector into a dense embedding vector that can be fed into the Simple RNN cells. The RNN layer takes the dense vector as well as the previous hidden state, and computes the next hidden state. When we reach the end of the sequence, the linear layer then takes the final hidden state and passes it to the fully connected layer, which is the output vector.

3.2 LSTM

A Recurrent Neural Network with Long Short Term Memory architecture takes a word embedding E that provides a vector representation for each of the words in the corpus. The LSTM model utilizes the LSTM memory cell, which has an input gate, output gate, forget gate, and cell memory. Each of these gates and cells has its own bias vector, and the hidden layer at each time step is a complex nonlinear combination of gate, cell, and hidden vectors.

The mathematical expression for the LSTM model is expressed as the following:

$$f_{(t)} = \sigma(W_{xf}x_{(t)} + W_{hf}h_{(t-1)} + b_f)$$
$$i_{(t)} = \sigma(W_{xi}x_{(t)} + W_{hi}h_{(t-1)} + b_i)$$
$$o_{(t)} = \sigma(W_{xo}x_{(t)} + W_{hi}h_{(t-1)} + b_o)$$
$$c_{(t)} = f_{(t)} * c_{(t-1)} + i_{(t)} * tanh(W_{xc}x_{(t)} + W_{hc}h_{(t-1)} + b_c)$$
$$h_{(t)} = o_{(t)} * tanh(c_{(t)})$$

The input gate, represented by $i_{(t)}$ controls which parts of the SimpleRNN term's output to add to the state. Then, $o_{(t)}$, the output gate, controls which parts of the cell state should be at the output. $f_{(t)}$, the forget gate, controls which parts of the cell state to forget. This architecture allows the model to control which parts of the sequence our LSTM will retain or forget.

We used a Python implementation of the LSTM model by Jonathan Raiman[5]. Following Graves (2014), we set the amount of LSTM inputs/outputs to be equal to the vocabulary size.

3.3 LSTM for Lyric Generation

The LSTM is advantageous in lyric generation in that it has the ability to remember long term dependencies, as well as longer sequences compared to other RNN models. As stated above, the LSTM has three gates that control the information it forgets, carries on in the sequence, and updates from the latest step.

3.4 CNN+LSTM

Since previous work has shown the power of LSTMs for lyric generation, we wanted to explore a more advanced model using two one-dimensional convolutional layers as feature extraction before feeding them into the LSTM, following the work by Zhou et al. (2015). This was in hopes of the CNN layers allowing the LSTM to learn certain features of an artist's style and help with the generation of the next word in the sequence.

Although the studies of Zhou et al. (2015) were done for text classification, their model was able to learn features through convolution layer, and sequences of such high level representations were then fed into the LSTM for long-term dependencies, which demonstrated the model's effectiveness in text classification.

[4]https://colab.research.google.com/

[5]https://github.com/JonathanRaiman/theano$_lstm/blob/master/Tutorial.ip$

3.5 Generating Lyrics

After training, the outputs were used to generate lyrics. Instead of using word vectors, simple base lyrics were created using a Python libray called Markovify[6]. This library was used to build Markov models based on the input text which generated seed sentence for lyric generation. For the purpose of our project, the seed sentence length was set to 3.

Once the seed sentences were obtained, the words were fed into each of the models, and then the lyrics were generated using the model. Then, the generated lyrics were rated against the given artist's lyrics with rhyme index and readability as the metrics. The output were 20 lines of lyrics and their average rhyme density and the average readability scores.

4 Experiments

4.1 Dataset

We used a pre-existing data set [7] from Kaggle that contained a collection of lyrics from 37 prominent rap artists. The developer of this data set has aggregated songs from the site genius.com. Our data set has been hand-cleaned and pooled so that all of a given artists' songs are in one .txt file where each verse is a separate line. The data set consists of 4 869 272 words, and 962 279 verses. For training, we used a .txt file as input, which includes a single artist's body of work and split it into two parts, one part for training and another for testing.

One observation to be made with regards to the data set is that not all .txt files were of the same length; some artists had released less songs compared to other artists and had a smaller set of lyrics.

4.2 Data Preparation

The nature of Ghostwriting is such that the ghostwriter must emulate the original artist's style, while creating novel content. To do this, we will train both the RNN, and LSTM models on one artist's body of work. We will generate words using the LSTM model with a word embedding layer.

For data preparation, we had to split the data into a smaller sequence sizes such that our models would be able to take a group of words as input to predict and generate another group of words as output. Then, these chunks of lyrics were tokenized and padded for equal lengths of sequences. The tokenized and padded sequences were then used as inputs for each of the models.

4.3 Baseline Implementation

Following the work inspired by Potash et al. (2018), for the baseline method, we are implementing a simple RNN model. To train the model we input a .txt file, which is an aggregation of lyrics from a single artist, to preserve their style as much as possible. Then, the RNN model generates a verse consisting of up to 10 words. We analyze the quality of these generated lyrics using our rhyme density evaluation metric.

The simple RNN model has the architecture of Input $\rightarrow$ Embedding $\rightarrow$ SimpleRNN $\rightarrow$ Output layer. The model utilizes the Adam optimizer with a learning rate of 0.001, and categorical cross-entropy for loss function. The number of epochs were set to 50.

4.4 LSTM Implementation

We use the same input data for the LSTM as the baseline method. The LSTM model has the architecture of Input $\rightarrow$ Embedding $\rightarrow$ bidirectional LSTM $\rightarrow$ Output layer.

We trained the model with bidirectional LSTM and Adam optimizer with learning rate of 0.001, and we will use cross entropy loss function to evaluate the loss function. The number of epochs were set to 120.

4.5 CNN+LSTM Implementation

The advanced model for LSTM inspired by the work of Zhou et al. (2015) was implemented. The architecture of this model is as follows: Input $\rightarrow$ Embedding $\rightarrow$ Convolution Layer $\rightarrow$ Convolution Layer $\rightarrow$ Bidirectional LSTM $\rightarrow$ Ouptut layer

Similar to the baseline and the LSTM, this model also took the tokenized and padded data as input, with the shape of training data as dimensions, and dropout rate of 0.2, along with Adam optimizer and learning rate of 0.001. This model had the number of epochs set to 220 due to its complex architecture.

However, despite using other tools such as Google Colab for GPU bandwidth, we had trouble generating lyrics with the advanced LSTM model, which was the combination of CNN and LSTM architecture. It took over 3 hours just to train the model, and the system kept crashing while generating lyrics. As a result, we had to compare our results from the baseline and the LSTM models.

[6] https://pypi.org/project/markovify/

[7] https://github.com/fpaupier/RapLyrics-Scraper/tree/master/lyrics$_U S$

5 Evaluation Metrics

The goal of automatic ghostwriting is to create a system that can take a given artist's work as input and generate similar yet unique lyrics. The evaluation metrics we used to determine the success of this project followed the manual evaluation and automated evaluation methods used by Potash et al. (2018).

5.1 Manual Evaluation Methods

The manual methods include the following criteria: evaluating the fluency and coherence, as well as the style matching of the lyrics. These are more subjective and may vary depending on the reviewer, so we reviewed the generated lyrics that will give scores based on the lyrics' fluency, coherence, and whether the style resembles that of the original artist.

Following Potash et al. (2018), fluency was measured by labeling each set of lyrics as "strongly fluent", "weakly fluent", and "not fluent" as numeric values 1, 0.5, and 0 respectively. Coherence was measured in a similar manner, where each set of lyrics were labelled "strongly coherent", "weakly coherent", and "not coherent" as numeric values of 1, 0.5, and 0.

5.2 Automated Evaluation Methods

The automated methods, on the other hand, aim to measure the uniqueness and stylistic similarity via rhyme density. Originally, we had planned to measure the uniqueness of generated lyrics using the if-tdf representation for verses along with the cosine similarity. However, through experiment, we concluded that the two metrics we selected, rhyme density and readability gave us more consistent results that was best suited for comparison of the two models we had implemented.

Rhyme Density: The rhyme density was calculated using the python library called Pronouncing[8], which counts the number of syllables and rhymes in a given text. The rhyme density was calculated by dividing the number of rhymed syllables by the total syllables.

Rhyme density gave us a quantitative measure of how similar the model's output is to the artist's work. A higher rhyme density indicates higher accuracy for the LSTM model. Other work by Xue et al. (2021) also suggests that rhymes are the key to form a good rap flow, which suggests that these are reasonable evaluation metrics for the generated lyrics.

Readability: Readability was calculated using another python library called Textstat[9], which had a built-in function for automatic readability score. We used the

$$\text{textstat.automated\_readability\_index}(\text{text})$$

function from Textsat which outputs a number that approximates the grade level needed to comprehend the text. The higher the number, the more complex the generated lyric is.

Readability measures the complexity of text by focusing on textual content such as lexical, semantical, syntactical and discourse cohesion analysis. It is computed in a very approximate manner, using average sentence length over words and average word length over characters in sentences.

6 Results and Analysis

Due to time constraints, only 12 artists were used to train and generate lyrics out of the 37 artists that we had originally obtained.

6.1 Example of Generated Lyrics

At first, the lyrics generated were very repetitive. This is an example of one of the first lyrics generated for Drake using the baseline model:

> Line 1: Please, please do me girl true yeah yeah yeah
> Line 2: For all my people back here tonight again woo
> Line 3: You send shots, though game play it too life some
> Line 4: But then got everything my name yeah
> Line 5: They don't really play it yeah yeah yeah yeah
> Line 6: Wow, I'm honored some you die before
> Line 7: You love me though thing yeah yeah yeah yeah yeah
> Line 8: She knew that time i— everything say

It can be observed that there are a lot of repetitive words in this set of lyrics. For example, the word "yeah" appears 13 times in this verse, and the word

[8] https://pypi.org/project/pronouncing/

[9] https://pypi.org/project/textstat/

”please” is repeated a few times as well. At this stage, it was difficult to determine whether this was caused by the model design, lack of data, or the artist’s style.

Another set of lyrics generated for Eminem using the baseline model, on the other hand, appears to be more complex.

> Line 1: In my shoes, daughter that pain now famous
> Line 2: I’ve tried in tomorrow girl me window
> Line 3: And maybe shout again me today tapped
> Line 4: In my shoes, daughter that pain now famous
> Line 5: I was daydreamin’ superman go on me sick of talkin’
> Line 6: I know there forever alright sad yeah yeah yeah
> Line 7: I never meant superman found you down
> Line 8: Didn’t I give a criminal world again oh sayin’

Although there are some repetitions, the words are not as repetitive as the lyrics generated for Drake.

6.2 Quantitative Analysis

The quantitative results were measured using the automated evaluation methods as outlined above.

Table 1 shows the rhyme density and readability index for the 12 artists we generated the lyrics with. From the table, in most cases, it can be observed that LSTM returns better scores for rhyme density and readability compared to the baseline method.

However, there were some exceptions. In the case of Eminem, Isaiah Rashad, and Mac Miller the rhyme density was higher in the baseline model. For readability, Ice Cube, Kanye West, and Kendrick Lamar had a higher score for the baseline model.

From this, we can conclude that the LSTM model generates better lyrics compared to the Simple RNN in terms of automatic evaluation metrics.

6.3 Qualitative Analysis

The fluency and coherence of each artists were measured manually. This was necessary because the rhyme density and readability metrics solely depended on the words in relation to each other within the given lyrics. In order for us as humans to evaluate the generated lyrics, another way to analyze the output was needed.

In addition, we discovered that higher rhyme density or readability did not necessarily correlate with better fluency and coherence.

This is Tupac’s lyrics generated using the LSTM model. Among the lyrics generated, it has the highest rhyme density score of 0.530.

> Line 1: Do it like that strong girl stressed like
> Line 2: I guess you snitches em all accepted that
> Line 3: For the life baby might death row creep back care
> Line 4: and who meee confused it up night me tonight here
> Line 5: No guts no home and my crew beeee superior caught
> Line 6: I’m faced with pimpin dizzy baby burn
> Line 7: and still stranded, on it light you there
> Line 8: Now rule one level tuff got style days blast blast

we can observe that there is some sort of rhyme scheme, with words that rhyme (care, here, there), and (that, caught, blast). However, if you look at the lyrics line by line, they seem somewhat fluent but not very coherent. Using the manual evaluation metrics mentioned above, this piece would be labelled ”weakly fluent” and ”not coherent.”

6.4 Limitations

Originally, in the beginning of the project, we used Google Colab for project setup, but the data had to be loaded every time we had logged out of the system. As a result, we had to move the work to our local machines for testing and troubleshooting, and run the completed code in Google Colab for faster training environment.

In addition, our models were not complex enough to generate lyrics that actually made sense semantically. We also noticed that generating lyrics multiple times using the same model resulted in lower rhyme density and readability. Since it took several hours for our models to generate data, due to time constraints, we were not able to explore much with changing the model architecture and optimizing them.

Artist	Baseline		LSTM	
	Rhyme Density	Readability	Rhyme Density	Readability
ASAP Rocky	0.413	3.669	0.478	3.625
Big L	0.405	2.6	0.414	2.605
Childish Gambino	0.396	2.565	0.444	2.649
Drake	0.391	2.375	0.423	2.399
Eminem	0.462	3.039	0.376	3.06
Ice Cube	0.254	2.644	0.365	2.61
Isaiah Rashad	0.411	2.62	0.322	2.96
J Cole	0.241	2.395	0.395	2.405
Kanye West	0.349	2.85	0.395	2.759
Kendrick Lamar	0.356	3.484	0.465	2.644
Mac Miller	0.327	2.93	0.305	3.055
Tupac	0.465	2.599	0.530	2.639

Table 1: Quantitative Results for Baseline and LSTM

7 Conclusion

Comparing the unsupervised, non-template LSTM model to a SimpleRNN, the LSTM model proved to be more effective for lyric generation. The SimpleRNN produced more incoherent lyrics and was objectively scoring lower in our readability and rhyme density metrics.

The lyrics generated by the LSTM model were not dramatically more coherent, however, as we are dealing with a musical genre that does allow for much poetic liberties, one would be hard pressed to judge an AI-generated lyric as being more obtuse than a given human-written lyrics. Although we did our best to define some evaluation metrics, we are in a very abstract realm and what "sounds good" to one ear might sound like nonsense to another, especially without music to accompany the lyrics. In the future, we would benefit from considering different evaluation metrics or exploring different genres of music to see how our model performs in different scenarios.

We must remember that the points of ghost writing is to spark creativity, not replace the artist, and sometimes creativity can come from incoherent or abstract places. Part of the compromises we had to make was around improving the model by increasing the data while keeping the program small enough so that our systems would not crash during execution. As such, we are hesitant to propose incorporating more data to train thee model as a means of optimization.

7.1 Future Work

For future work, we would like to update the LSTM model to generate better lyrics that resemble human work. Some methods that could be applied are changing the parameters, the model architecture, or the embedding dimensions or hidden dimensions.

Another method would be to optimize the CNN+LSTM model as explored by Zhou et al. (2015) which was the more advanced LSTM model we had attempted to implement.

7.2 Contributions

Both group members worked together in planning and discussing the project details as well as writing the report.

Grace worked on the LSTM model setup and training, generating lyrics, and evaluation of the models. Grace also created the slides and outlined the key points for the project presentation.

Tara gathered the data required for the project, completed the data preparation step, and set up and trained the baseline model. Also, Tara recorded the video presentation.

References

Alex Graves. 2014. Generating sequences with recurrent neural networks.

Eric Malmi, Pyry Takala, Hannu Toivonen, Tapani Raiko, and Aristides Gionis. 2016. Dopelearning. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Nikola I. Nikolov, Eric Malmi, Curtis Northcutt, and Loreto Parisi. 2020. Rapformer: Conditional rap

lyrics generation with denoising autoencoders. In *Proceedings of the 13th International Conference on Natural Language Generation*, pages 360–373, Dublin, Ireland. Association for Computational Linguistics.

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, Lisbon, Portugal. Association for Computational Linguistics.

Peter Potash, Alexey Romanov, and Anna Rumshisky. 2018. Evaluating creative language generation: The case of rap lyric ghostwriting. In *Proceedings of the Second Workshop on Stylistic Variation*, pages 29–38, New Orleans. Association for Computational Linguistics.

Alex Sherstinsky. 2020. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306.

Lanqing Xue, Kaitao Song, Duocai Wu, Xu Tan, Nevin L. Zhang, Tao Qin, Wei-Qiang Zhang, and Tie-Yan Liu. 2021. DeepRapper: Neural rap generation with rhyme and rhythm modeling. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 69–81, Online. Association for Computational Linguistics.

Chunting Zhou, Chonglin Sun, Zhiyuan Liu, and Francis C. M. Lau. 2015. A c-lstm neural network for text classification.