Rap Lyrics Generation with Generative Adversarial Networks

Chengfan Li
University of Michigan
Ann Arbor, MI, USA
lllllcf@umich.edu

Luyang Hu
University of Michigan
Ann Arbor, MI, USA
hulu@umich.edu

Shaochun Zheng
University of Michigan
Ann Arbor, MI, USA
zhengsc@umich.edu

I. Introduction

As rap music quickly evolve into a mainstream musics genre, it has triggered the interest of a number of linguists and computer scientists. Rap lyrics are often considered one of the most complex kind of lyrics. Eligible rap lyrics require not only creativity to tell an intriguing story but also lyrical skills to accommodate the strict rhyme patterns. Our project aims to study the computational creation of rap lyrics and make everyone able to "write" their own lyrics only by entering a piece of phrase. By utilizing multiple NLP techniques, we hope to generate rap lyrics that models the meaning, the rhyme, and the rhythm. Below is an example output with input phrase "Disability divulge":

> **Disability divulge** *mathematics tulip, apoplexy speedily.*
> *Powerful, slightly politic inventiveness especially.*

All the codes and data of this project are available on Github. [1]

A. Overview

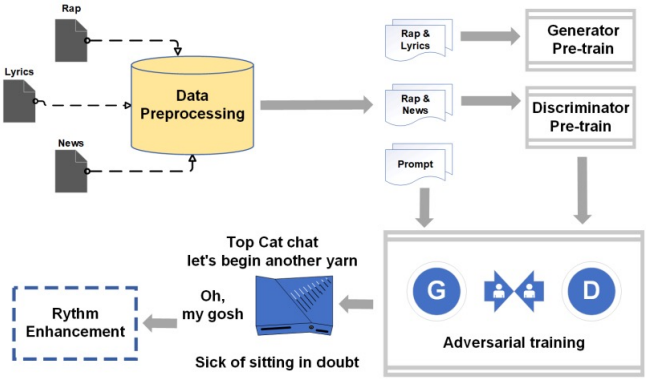

Fig. 1. Pipeline Demonstration

As shown in Fig. 1, our model includes a generator, a discriminator, and a generative adversarial network(GAN). The generative model is trained by the rap lyrics and the music lyrics we gathered, while the discriminative model is trained by the rap lyrics and arbitrary news texts. The human input prompt will go through adversarial training to generate an elementary rap lyrics. Lastly, rhyme enhancement module makes sure that the rhyme at the end of sentences always follow the pattern of the previous ones. More details on each part of the design is discussed later.

[1]https://github.com/lllllcf/487-Project-Rap-Lyrics-Generation

B. Challenge

There are two major challenges in our project. First, it is hard to encode both semantic, rhyme, and rhythm all together, especially for rhyme and rhythm. Rappers change their tones and pronunciations frequently, so the normal phonology translation does not work ideally. Also, rappers often adjust their rhythm by the beat, which is difficult to capture solely from rap lyrics without the sound track. Another main challenge here is the cost of training. Adversarial network takes use several hours to train without interruption. It is very time-consuming when adjusting the parameters or modifying model.

C. Contribution

Chengfan: Design and implement the GAN model; run tests and adjust parameters

Luyang: Look over related works; test and modify model from those works to help ours

Shaochun: Gather and clean the data; build rhyme enhancement module

II. Related Works

A. Previous Works

One of the most important part of this project is to figure out how to capture and properly encode/decode the rap lyrics. Our goal is to transform the vector by semantic embedding, phonetic features, and hopefully the rhythm features. Based on existing publications, we decided to test our methods on mandarin first from DeepRapper [1], a great model for Chinese rap lyrics generation from Microsoft Team project muzic. Their encoding strategy is to concatenate feature vectors into one (shown in Fig. 2). To capture rhyme, they used the vowel embedding together with the relative position

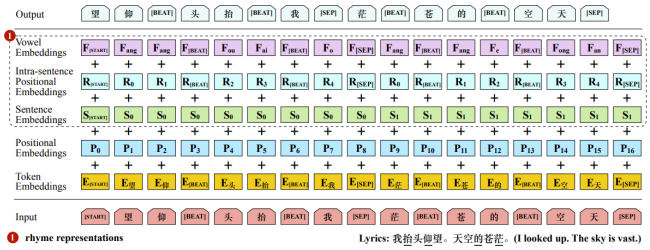

Fig. 2. Visualization of the DeepRapper architecture

vector to decide the correspondence between the rhyming words in consecutive sentences. For the rhythm feature, they includes token [BEAT] in the rap lyrics data, which is extracted by their data processing pipeline from the sound track. For the semantic, it is straight-forward to use sentence embedding.

We ran the implementation of Deeprapper and generate a chunk of lyrics using the pre-trained model they provide. It turns out their generation is quite reasonable: [B]/[BEAT] stands for a stop or a punctuation.

我长大的地方像个简朴的寨
有种莹莹的彩
有 [B] 一个姑娘 [B]
我 [B] 不 [B] 用再跟你听话
吃 [B] 最简朴 [B] 的菜
这里有着简朴的爱
一份纯 [B] 朴 [B] 的朴 [B] 菜
伴 [B] 随着 [B] 节奏 [B] 跳 [B] 起来
[B] 任 [B] 由 [B] 又自 [B] 在
[B] 让 [B] 我唯美 [B] 的 [B] 期 [B] 待
喜 [B] 欢耍 [B] 的孩 [B] 儿不坏
[B] 一 [B] 杯纯 [B] 白的 [B] 喊麦

To further extend the encoding strategy to English rap songs, which is our goal, we referenced another project, DopeLearning [2]. It characterizes rhyme into many types, like perfect rhyme, alliteration, and consonance, while to most common only is assonance. It simply means the vowel are shared, like 'crazy' and 'baby'. Other categories, for instance, perfect rhyme is more strict. The words share exactly the same end sound, like 'slang-gang' . The rhyme contained in English rap seems to be far complex than we originally thought, and we decided to focus on assonance.

B. Improvements

Our model uses a GAN model. Unlike the pure generative models or style transfer models they used, we add the step to distinguish the rap lyrics from other genre. Also, we transferred the methodology from Mandarin rap to English rap using completely data and encode/decode technique. Finally, we built Rhyme Enhancement Module on BERT to ensure the perfect rhyme alignment at the end of sentence.

III. Data preprocessing

In this section, details of data collection and cleaning will be discussed, as shown on Fig. III-C2. The data split detail will be provided on Experiment section.

A. Data Collection

We collected three sets of data for this project. The first set of data is a table of news articles downloaded from the database Kaggle [3], whose content ranges from politics and economy to science and technology. It is used as the negative samples for the discriminator, as we need to train the discriminator to distinguish rap lyrics from other texts. The second set of data is the lyrics of songs of all genres, collected from the same database as mentioned beforehand. Since the method of transfer learning is employed, we pre-train our model on the generic lyrics to generalize the feature extraction. The third set of data is the rap lyrics, collected from Kaggle [4] and OHHLA [5], on which we will fine-tune our model. The size of the data that we collected is show on Table I.

TABLE I
Dataset Size

	Dataset	Size
Rap Lyrics	Kaggle & OHHLA	172,866 sentences
Song Lyrics	Kaggle	172,866 sentences
News	Kaggle	132,626 sentences

B. Data Cleaning

In data cleaning part, we first lemmatize each word to condense text information. Then, we remove numbers, names and punctuation that may affect the efficiency of word embedding.

Initially, stop words are removed, but later we found that this would make the generated lyrics grammarly incomplete. Finally, we decide to keep these words.

C. Text Representation

Then we further transfer our data into vector to fit into the structure of our model.

1) Generator Input Embedding: For the input of the generator, we use Word2Vec to map words to vectors. The reason for this is that we need to use vectors to represent the meaning of words and try to get the machine to understand and generate completely new lyrics.

Suppose that we have a list of prompt words W with size l and Word2Vec model M, then the input of generator is $M(W)$ which is a $b \times l \times \text{len}(M(\cdot))$ tensor with batch size b. And the output of the generator is a $1 \times \text{len}(M(\cdot))$ vector for the prediction of next word.

2) Discriminator Input Embedding: For the input of the discriminator, We first use Word2Vec to map the word to vector. Since the aim of discriminator is to identify whether the current sentence is real rap lyrics, and each word in a sentence has a different weight, we use tf-idf to describe the importance of the current word. The final sentence representation is obtained after weighted averaging based on word embedding and tf-idf.

Suppose that we have a list of words W with size l, Word2Vec model M, tf-idf function T, then the sentence embedding of the input of the discriminator is,

$$\text{Input} = \frac{1}{l} \cdot \sum_{i=0}^{l} M(W[i]) \cdot T(W[i])$$

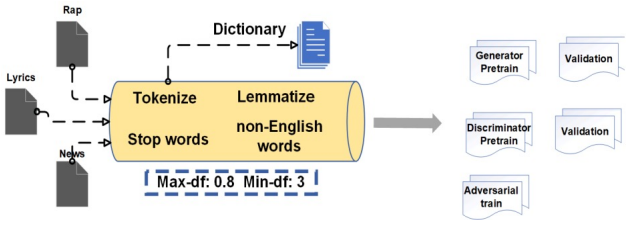

Fig. 3. Data Preprocessing Pipeline

IV. Methodology

In this section, the steps and model architecture, details and ideas of rap lyrics generation are discussed, including GAN and Rhyme Enhancement. Since this is a generative task, it is difficult to train and evaluate the generator alone. The use of existing evaluation metrics may not fully reflect the advantages and disadvantages of the generator, nor can it perfectly fit out target. So in this case, we introduce the discriminator and jointly train the two to get a better and more unique generator. Moreover, since the generative model may not capture the rhyming feature of rap lyrics, we additionally apply rhyme enhancement to the output.

A. GAN [6]

1) Architecture: As shown on Fig. 4, the generator contains one embedding layer, one LSTM [7] with two layers and one fully connected layer. As discussed on the Text Representation section, We feed the embedding layer of the generator n word embedding vectors and use its output as input to LSTM to predict the next word. Then we calculate the mean square loss for the word embedding of generated words and the word embedding of the real next word.

As shown on Fig. 5, the discriminator is a simple binary classifier containing three fully-connected layers and one output indicating whether the input sentence is a rap lyrics. As discussed on the Text Representation section, the input of the discriminator is word embedding vectors weighted by tf-idf.

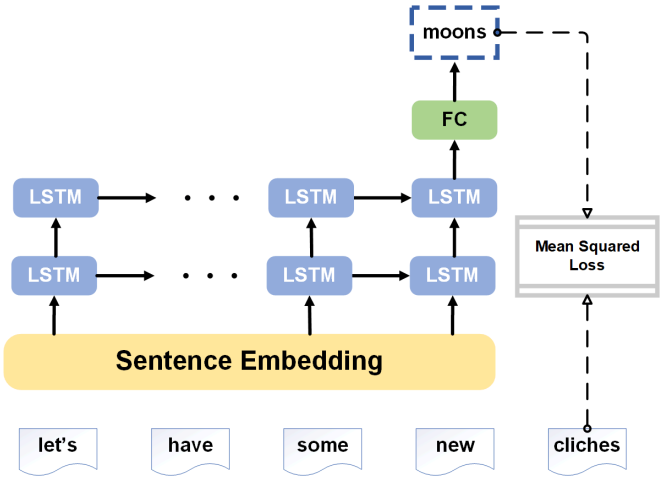

Fig. 4. Generator Structure

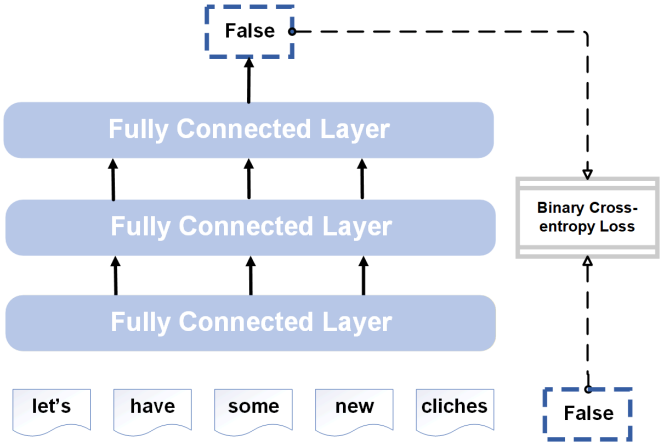

Fig. 5. Discriminator Structure

2) Pre-training: We first pre-train generator and discriminator to make them better against each other in the adversarial training instead of generating random meaningless results, but we cannot train both models too well in pre-training, doing so will lead to inconsistencies in adversarial-training. We use rap-lyrics to train the generator for 10 epochs and use rap and news data to train the discriminator for 5 epochs.

3) Adversarial-training: Next, we enter the adversarial training stage. The generator receives a short piece of lyrics as input, and generates a new piece of lyrics, which is then handed over to the discriminator to determine whether the text is a rap lyrics.

Suppose that $D(x)$ is the output of the discriminator that estimate the probability that the instance x is real. $G(z)$ is the output of the generator with input z.

The minimax loss is given by,

$$\text{Loss}_{\text{mm}} = \log[D(x)] + \log[1 - D(G(z))]$$

The goal of generator is to minimize the loss and the discriminator tries to maximize it. This loss function bet-

ter considers the consequences of the adversarial results between the two, and dynamically links them.

The loss is then back-propagated to update the parameters of generator and discriminator. Note that since the generator's output is connected to the discriminator's input at this stage, the discriminator parameters should not be updated when back-propagating the generator loss. Repeat this process, training both alternately until convergence.

B. Rhyme Enhancement

One of the most important feature of raps is rhyming. Nevertheless, as an underlying pattern, the rhyming scheme in the input data may not be necessarily caught by our generative model. Therefore, it is of importance to enhance the rhyming after the lyrics are generated. The steps of rhyme enhancement is shown on Fig. 6.

In order to make the lyrics rhyme with each other and preserve the semantic and syntactic information in the original sentences, we use the BERT [8] model to aid us with this task [8]. After fine-tuning BERT on the rap dataset, we mask the last several words of one sentence (the source sentence) and then use BERT to fill the mask. After a list of candidates are returned, we first give an original score based on the number of consecutive identical phonemes in the candidate and the word to rhyme with (the target). Considering the rhythms in raps, the original score is adjusted so that the difference between the lengths of sound of the source and target is penalized.

$$p_{\text{length}} = 1 - \frac{2}{\pi}\arctan[w_1(|\text{target}| - |\text{candidate}|)]$$

Moreover, the order in which the candidates are given by BERT is also taken into consideration by penalizing ones of a lower priority, which indicates less semantic and syntactic fitness.

$$p_{\text{priority}} = 1 - \frac{2}{\pi}\arctan(w_2 i)$$

In order to avoid the selected replacement being the same as the target word, a candidate is specially penalized if it is the same as the target word.

$$p_{\text{identity}} = \begin{cases} w_3 & \text{if candidate} = \text{target} \\ 1 & \text{otherwise} \end{cases}$$

The final score is the original score multiplies with the penalties.

$$s = s_0\, p_{\text{length}}\, p_{\text{priority}}\, p_{\text{identity}}$$

The candidate with the highest final score is selected and replaced for the original word. This method is conducted on every pair of sentences to achieve the rhyming effect.

All of the weights of the penalties as mentioned above are adjustable. Given the quality and feature of our current generative model, we decrease the weight of the penalty based on priority and increase the weight of the penalty based on length difference.

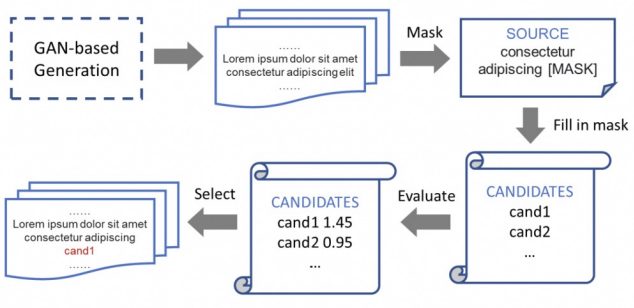

Fig. 6. Rhyme Enhancement Pipeline

V. Experiment

A. Data Split

We construct five data loaders in out model.

1) Generator pre-training loader and Generator pre-training validation loader
   These datasets contain 70 percent of the rap lyrics we collected and we replace 10% of the content with other kinds of song lyrics to ensure the generalization of the training set.
   The ratio of training and validation sets is 10:1.
2) Discriminator pre-training loader and Discriminator pre-training validation loader
   These dataset contains 30 percent of the rap lyrics we collected and equal amount of news.
   The ratio of training and validation sets is 10:1.
3) Final training loader
   This dataset contains lyrics, news, and randomly generated prompt sentences as input to the generator for adversarial training of generator and discriminator.

For final test of out model, we use real rap lyrics as prompts for generator to generate new rap lyrics.

B. Hyperparameter tuning

Due to the complex architecture and training process, GAN has a large number of hyperparameters that can be tuned.

Name	value	Description
gpre_epoch	20	generator pre-train epochs
dpre_epoch	8	discriminator pre-train epochs
pre_batch	64	pre-train batch size
s_n	10	pre-train input sequence length
dhidden_dim	512	discriminator hidden dimension
gl	2	LSTM layers
ginput_s	300	LSTM input size
ghid_dim	512	LSTM hidden dimension
optim	adam	optimizer
lr	0.001	learning rate
wd	0.0001	weight decay
epoch	50	final training epochs

Moreover, we use mean square loss for pre-training Generator, binary cross entropy loss for pre-training Discriminator and minimax loss for adversarial training stage.

C. Pre-train Result

Fig. 7 shows the pre-train result of generator. We train the generator for 20 epoches to make sure that the generator can generate some basic sentences instead of randomly generating noise. On the graph we can observe that the validation set loss is still decreasing, which means that we are not training the generator to the best possible state in pre-training, this is because continuing to train more will only make the generator 'remember' all the sentences and data in the training set to fool the discriminator, it is a better choice to stop at an earlier stage and hand over the rest of the training to the discriminator.

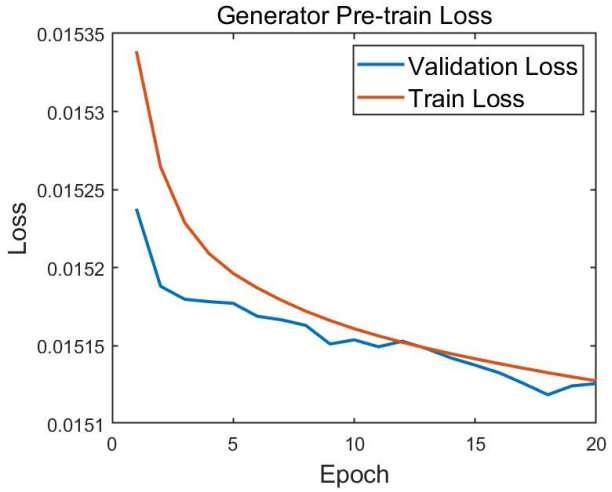

Fig. 7. Generator Pre-train Result

Fig. 8 shows the pre-train result of discriminator. We can observe that the discriminator reach a good accuracy on an early stage.

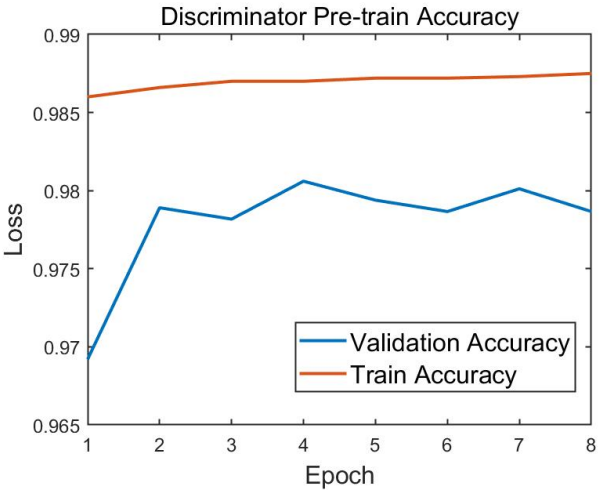

Fig. 8. Discriminator Pre-train Result

D. Adversarial Training Result

Then, we enter the adversarial training stage. In this stage, we pass a batch of data to the generator to generate new rap lyrics, and then use the discriminator to identify whether the generated lyrics are rap. After calculating the loss generated by the current batch, back-propagation is used to update the parameters. The rap lyrics are regenerated using the finally obtained generator.

After another 50 epochs of adversarial training, our generator generated the following lyrics.

Elusive painter dutch allege radical forest
Distressed peyote lend igloo soccer.
Altruistic do deference forsaken Michigan.
Disability divulge mathematics tulip, apoplexy speedily.
Greenery retardant verbalize courthouse grammatical.
Powerful, slightly politic inventiveness texture.

On Evaluation section, we will further evaluate these generated lyrics based on several metrics.

E. Rhyme Enhancement Result

Applying rhyme enhancement to the generated lyrics, we obtain the following results:

Elusive painter dutch allege radical ***culture.***
Distressed peyote lend igloo ***soccer.***
Altruistic do deference forsaken ***only.***
Disability divulge mathematics tulip, apoplexy ***speedily.***
Greenery retardant verbalize courthouse ***grammatical.***
Greenery retardant verbalize courthouse ***signature.***

It can be observed that the last word of each sentence is adjusted so as to rhyme with each other.

F. Evaluation

The two most important characteristics for rap lyrics are their rhyming level and readability. To further check the quality of our generated lyrics, we introduce the following two evaluation metrics.

1) Rhyme Density [2]
   Rhyme Density quantify the quality of lyrics from a rhyming perspective and measure average length of the longest rhyme per word.
   After Rhyme Enhancement, the Rhyme Density of our generated rap lyrics is 0.65 while the Rhyme Density of the rap written by The Lonely Island is 0.87 [2].
2) Readability We use Smog Index to measure the readability of the generated rap lyrics. Smog Index is a grading formula that a score of 9 means that a ninth-grader would be able to read the document and it can be calculated by,

$$\mathrm{S} = 1.043\sqrt{\#\text{polysyllables} \times \frac{30}{\#\text{sentences}}} + 3.1219$$

   The Smog Index of our generated rap lyrics is 11.2.

G. Baseline Model

We choose the language model as our baseline, which generates the following results,

> *Elusive painter , i 'm a superstar , made it 's the questions (what ?) , you know.*
> *but i 'm a superstar , made it 's the questions (what ?) , you know i 'm.*
> *told you that i 'm a superstar , made it 's the questions (what ?) , you know.*

We can see that the lyrics generated by baseline are simpler, but full of repetitions, such as 'I am a super star', 'you know' and so on.

The Rhyme Density of the rap generated by baseline model is 0.33 and the Smog Index is 8.1 which consistent with our analysis above.

VI. Discussion

In the beginning, we tried to map words to index and then give them to the generator and discriminator for training, but we later found that this did not train the meaning of words well, so we later represented the input as word embedding.

Although the generated lyrics have a certain degree of readability and meaning, the syntax and semantics are still relatively rigid and difficult to understand, since the training set contains many uncommon words and phrases, the generated content is not filtered during the generation and the model is still undertrained. Moreover, there may be problems with word embedding and input selection. Better embedding methods can be investigated to further improve our model.

References

[1] L. Xue, K. Song, D. Wu, X. Tan, N. L. Zhang, T. Qin, W.-Q. Zhang, and T.-Y. Liu, "Deeprapper: Neural rap generation with rhyme and rhythm modeling," *arXiv preprint arXiv:2107.01875*, 2021.

[2] E. Malmi, P. Takala, H. Toivonen, T. Raiko, and A. Gionis, "Dopelearning: A computational approach to rap lyrics generation," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016, pp. 195–204.

[3] A. Thompson, "All the news," https://www.kaggle.com/datasets/snapcrack/all-the-news, 2017, [Online; accessed 25-Mar-2022].

[4] X. Song, "Hip-Hop Encounters Data Science," https://www.kaggle.com/datasets/rikdifos/rap-lyrics, 2021, [Online; accessed 25-Mar-2022].

[5] "Hip-Hop & Rap Lyrics at OHHLA.com," https://ohhla.com/all.html, 2022, [Online; accessed 25-Mar-2022].

[6] D. Jin, Z. Jin, Z. Hu, O. Vechtomova, and R. Mihalcea, "Deep learning for text style transfer: A survey," *Computational Linguistics*, pp. 1–51, 2021.

[7] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.