Take My Ad-lib: Hip-hop Freestyle Generator Using Specific Key Words

Wang Xinghong, Weng Penghao, Sheng Tang

Abstract

Freestyle, a special way of performance of Hip-hop music, is an important characteristic of human impromptu show. In this project, we target on Hip-hop freestyle lyrics generation given specific user-defined keywords. Semantic, phonetic and keyword features are engineered for generation. A multi-encoder decoder language model is implemented with attention mechanism. Experiment results shows that our model successfully generates understandable hip-hop vibe lyrics related to given keywords which fulfill rhyme constraints.

I INTRODUCTION

Hip-hop music has been more and more popular as its occupation of the top positions of every Billboard ranking. Apart from the great musicality of rhythm and groove, a large proportion of the appeal of hip-hop music come from its lyrics. Hip-hop lyrics feature a frequent deployment of rhyme and improvising.

Particularly, there is a special way of performance under the genre of hip-hop music which is called ”freestyle”. Usually the performer is given a topic or a keyword, and he/she needs to improvise with lyrics that are somehow related to the provided topic or keyword. In this kind of performance, musicality is actually reduced and lyrics are enhanced.

In this project, we are going to design and implement a model which can generate hip-hop freestyle lyrics given a keyword. We will neglect the musicality and focus on the quality of the generated lyrics, making sure that rhyme is appropriate and the style is correspondent to that of typical hip-hop music.

II DATA PRE-PROCESSING

We mainly utilize a Kaggle dataset named Hip-Hop Encounters Data Science containing hip-hop lyrics from 36 famous rappers [1]. Lyrics are classified according to artist name, those from each rapper stored in a text file.

We interviewed some college students in Shanghai who actually make hip-hop music to ask how they do freestyle in real life given certain keywords. The answer is that they usually come up with lyrics sentence by sentence. The next sentence is based upon the content of the last sentence, the rhyme that they decide to set up, and the keywords given. Provided with such information, a sequence to sequence model is suitable for such a lyric generation task. Encoder-decoder architectures with attention mechanism have been gaining popularity these years for sequence to sequence task like machine translation these years [2]. So we decide to also utilize encoder-decoder-like model with attention mechanism to complete our task.

Three features are to be extracted to predict the next line of lyric: 1) the semantic content of this line of lyric, 2) the rhyme (phonemic feature) of this line, 3) keywords. For rhyme, in other words, phonemic feature extraction, we use CMUDict [3] in order to map last few words to their phonemic representations. After study of the lyrics and discussion with rap students with rap music production experience, we decide to focus on only the last 2 vowel phonemes of each sentence to make the rhyme feature extraction concise. Each verse, instead of each line, will be concluded into three keywords, for the prevention of inefficiency of keyword extraction from very short texts. The tool for keyword extraction is KeyBERT [4]. Verses are fed into the KeyBERT model and the results are returned and filtered with the masking of several frequent indecent words.

During the close look into the text files and the lyrics, we found some problematic and noisy lyrics and did some data cleaning on the following three types of low-quality lines. 1) Meaningless lyrics. For example, some verses are filled with consecutive ”ah ah ah”. 2) Highly repetitive lines which may lead to our final generated lyrics being repetitive as well. 3) Functional words and symbols such as ”[Rapper Name] * 4”. We wrote functions to detect and eliminate those verses that are overly occupied with those problematic lines.

After iteration of the text files and practice of the feature extraction and data cleaning work, we finally got 99938 tuples of information, each tuple containing five features, namely ”artist name”, ”this line”,”phoneme”,”keywords” and ”next line”. In the model design, we are going to utilize the ”this line”, ”phoneme” and ”keywords” to predict ”next line”.

III MODELING

We choose to design and implement an encoder-decoder architecture with attention mechanism to complete the task of hip-hop freestyle lyrics generation. We made attempts with both pretrained and un-pretrained models. Several controlled experiments were carried out for our reference to the final choice.

III.1 Encoder-Decoder Model Choice

In our first trial, we encode the three features: ”phoneme”, ”keywords” and ”this sentence” using three independent encoders and feed them into our decoder which has a transformer-like structure as shown in Fig.1. For phoneme feature, we take a transformer model [5] as its encoder, and for keyword and ”this sentence” feature, we take Bert Pretrained Model [6] (Bert-base-uncased) as their encoders. The output hidden states would feed into the un-pretrained decoder step by step using cross multi-head attention [7] with input of the decoder.

However, this model received a bad result with perplexity in the range from 88 to 98 and need tons of time to train it. A baseline encoder-decoder model with only ”this sentence” fed into the encoder was checked, and we still received a high perplexity. So, we decided to check a pre-trained language model as our decoder: GPT2 [8].

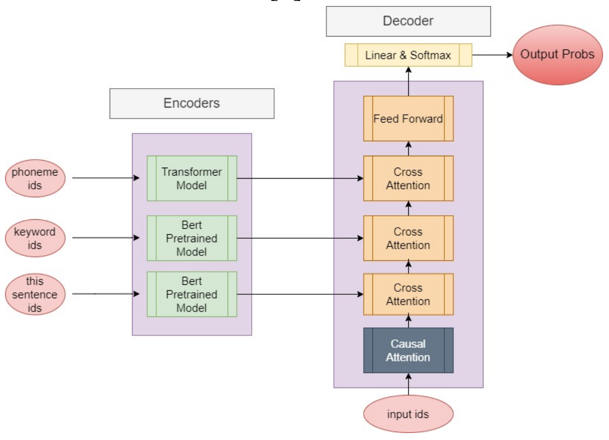


Figure 1: Transformer-like E-D Model

Two pretrained encoder-decoder model designs with only ”this sentence” as feature were implemented. One was BERT as the encoder and GPT2 as the decoder. The other was GPT2 as both the encoders and the decoder.

The Bert Encoders and GPT2 decoder design showed a relatively lower perplexity which was equal to 42. Although phoneme features and keywords features were not shown, we successfully generated some understandable lyrics with a typical hip-hop vibe. This provided a new direction for us, which was the utilization of both pretrained encoders and pretrained decoder.

Next step was to feature engineer the three features, combine them into a final hidden state to feed into the decoder.

Assume that the size of ”this sentence ids” is (B, M), then for ”this sentence” feature, we directly take its encoder output hidden state with size (B, M, H). For keyword feature, we take its [CLS] hidden state from the output as a representative tensor with size (B, 1, H), and repeat it M times to fit size (B, M, H) in order to make every token from this sentence able to learn the keyword feature. For phoneme feature, we embed phoneme’s hidden state with size (B, 2, H) for only the last two tokens in ”this sentence” to learn the phoneme feature, and fill other tokens with 0 in order to get a (B, M, H) size hidden state.

$$H_e^{final} = W_p H_e^p + W_k H_e^k + W_s H_e^s$$

Then we perform a linear projection and addition operation upon these three hidden states to reach a final hidden state for the encoder output, which can be feed into the Decoder layer for cross attention calculation. This model design is demonstrated as in figure 2.

A perplexity of 49 was achieved using this model design, which was higher than the one-feature encoding. We wanted to figure out what feature makes the perplexity higher than the one-feature Bert2GPT2 model, so we made two controlled experiments.

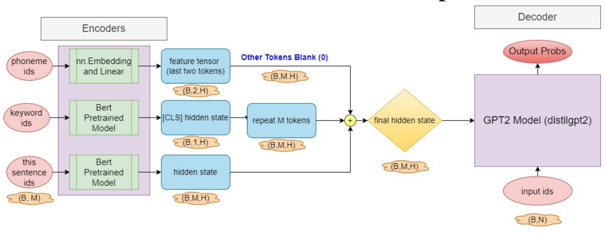


Figure 2: Three Features E-D Model

III.2 Feature Selection

In the first controlled experiment, we removed the keyword feature, and got a 45.9 perplexity, which is better than the three-feature model which was described in previous subsection. In the second experiment we took away the phoneme feature (Figure 3), and successfully reduced the perplexity to 41.8.

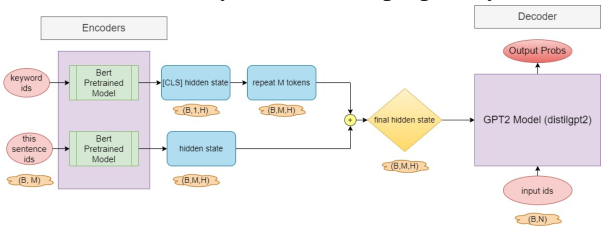


Figure 3: ”No Phoneme” Feature E-D Model

The ”no phoneme” model shows the best performance of all experimented models. Besides, keyword features seem to be successfully learned by the model and applied to the generated lyrics. For example, if one of the entered keyword is ”Religion”, content related to ”church”, ”love”, and ”Christian” often appear in the generated freestyle. As a result, we decide to take ”no phoneme” model as our final choice due to the learning results. Rhyme (phoneme) problems will be resolved during generation.

Below is the table of all the results from different models we tried. In this table, ”E-D” stands for Encoder-Decoder Model, ”base” stands for a baseline test, ”s” stands for ”this sentence” feature, ”k” stands for ”keyword” feature and ”p” stands for ”phoneme” feature.

Model	Features	Perplexity
Transformer-like E-D	s,k,p	98
Transformer-like E-D base	s	88
Bert2GPT2 E-D base	s	42
GPT2GPT2 E-D base	s	50
Bert2GPT2 E-D	s,k,p	50
Bert2GPT2 E-D	s,k	41.8
Bert2GPT2 E-D	s,p	45.9

III.3 Modeling Summary

To summarize the modeling process, we: 1) choose the Bert pretrained Model as Encoders and the GPT2 pretrained Model as Decoder, 2) take two features: ”keyword” and ”this sentence” to feed into the Encoders, 3) feature engineer the hidden states from the two Encoders and get a final hidden state to feed into the Decoder layer, 4) use the cross-attention mechanism to feed the encoder hidden states into the decoder 5) solve the rhyme problem during the generating process, details in Section IV.

IV LYRICS GENERATION

IV.1 Traditional beam search

One classical way to reduce the time complexity of decoding is to use beam search. Traditional beam search is basically an extension of greedy algorithm, which is intended to decode the sentence with highest. If we can choose beam search wisely, we can significantly reduce the decoding time and get a reasonable output. However, there are some problems occurred when we apply traditional beam search to our model. So we decide to modify the original beam search to reach our expectation of our lyrics.

IV.2 Modified beam search

We introduce a modified beam search due to the following reasons:

- Traditional beam search simply concerns about the fluency of the output, however, we need to concern the rhyme of the output lyrics.
- Traditional beam search will easily generate in-sentence repeated trigrams.

- In our case, traditional beam search will also generate sentences with the same popular n-grams.

To solve the first problem mentioned above, we add phoneme constrains when generating next sentence. Since our phoneme feature is merely a two-dimensional vector with values ranging from 0 to 16, we use dot product to calculate the similarity between phoneme features between two sentences. The formula below is to convert similarity to score:

$$S_{Phoneme} = log(v_1 \cdot v_2)$$

$$S_{Final} = S_{Model} + S_{Phoneme}$$

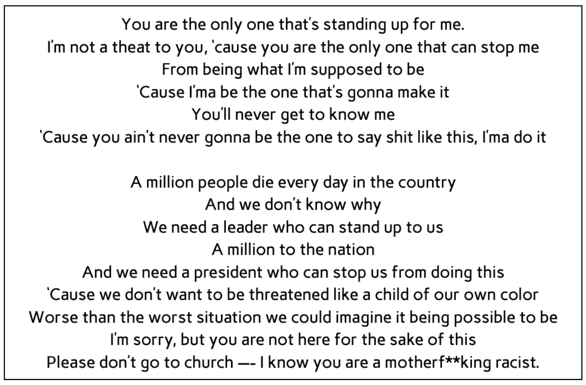
And at the end of the beam search process, we calculate the new scores for all the candidates trying to find a new sentence with highest S_{Final}. Solving problem 2 and problem requires similar trick, so we introduce them together below. To prevent repeated popular n-grams, either in-sentence or cross-sentence, we prune the paths we don't want before we calculate new scores for all candidates.

- Pruning sentences with too many popular n-grams helps our model generate reasonable sentences.

- During generation, there are some n-gram pairs acting like a 'trap'. For instance, let's assume a sentence with n-gram A is very likely to generate a new sentence with n-gram B. However, if the new sentence is very likely to generate a new sentence with n-gram B, then our model can only generate sentences with A and B. Pruning sentences having the same popular n-grams as previous sentence can help our model escape from the 'trap'.

V RESULT & FUTURE WORKS

In this project, we managed to produce some reasonable and understandable hip-hop vibe lyrics corresponding to the input keywords which fulfill rhyme constraint provided by previous lines of lyrics. For example, see Figure 4, which is a re-formatted version (e.g. manually break one line into two to make it look neat) of our original sample output. However, there are still things we could do to improve our model in the future.


You are the only one that's standing up for me.
I'm not a theat to you, 'cause you are the only one that can stop me
From being what I'm supposed to be
'Cause I'ma be the one that's gonna make it
You'll never get to know me
'Cause you ain't never gonna be the one to say shit like this, I'ma do it

A million people die every day in the country
And we don't know why
We need a leader who can stand up to us
A million to the nation
And we need a president who can stop us from doing this
'Cause we don't want to be threatened like a child of our own color
Worse than the worst situation we could imagine it being possible to be
I'm sorry, but you are not here for the sake of this
Please don't go to church —- I know you are a motherf**king racist.

Figure 4: Keywords:Religion Race

The first possible improvement for our model is about the phoneme feature. We add phoneme constrains during the generation process, instead of directly letting the language model to learn about phoneme feature. This makes the rhyme seems a little stiff. In the future work, adding the phoneme feature to language model itself would be an essential task.

The second possible way is finding a more opportune Encoder-Decoder model for our generating task. In our initial envisage, we wanted to make three cross attention in the decoder layer for the three different features. However, we only managed to achieve two features from the encoders. A more opportune model could provide lower perplexity and generate more understandable lyrics in our anticipation. So, ameliorating the model in this approach would still be a great challenge in the future work.

References

[1] Seanny, *Hip-Hop Encounters Data Science*, Kaggle, 2021. Accessed on: May 1, 2022. *[Online]. Available: https://www.kaggle.com/datasets/rikdifos/rap-lyrics.*

[2] K. Cho, B. Van Merrienboer, C. Gulcehre, D. Bahdanau, ¨ F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078, 2014.*

[3] Carnegie Mellon Speech Group, *the Carnegie Mellon Pronouncing Dictionary,* Pittsburgh, Pennsylvania, Carnegie Mellon Univercity, Copyright (C) 1993-2014.

Access on: Abbrev. Apr. 25th, 2022. Available:https://github.com/cmusphinx/cmudict

[4] M. Grootendorst, “Keybert: Minimal keyword extraction with bert,” *[Internet]. Available: https://maartengr. github. io/KeyBERT/index. html,* 2020.

[5] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, et al., “Huggingface’s transformers: State-of-the-art natural language processing,” *arXiv preprint arXiv:1910.03771,* 2019

[6] 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

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” *Advances in neural information processing systems,* vol. 30, 20

[8] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al., “Language models are unsupervised multitask learners,” *OpenAI blog,* vol. 1, no. 8, p. 9, 201