Lyric Generation with Style

Yuxi Feng
The University of British Columbia
fyx14@cs.ubc.ca

Bicheng Xu
The University of British Columbia
bichengx@cs.ubc.ca

Abstract

Text generation is a popular task in the natural language processing area. However, it remains challenging for generative models to generate coherent texts. These years, generative adversarial network (GAN) has shown an amazing performance in image generation and thus recently, many people began to apply GAN for text generation. In this work, we propose a GAN-like framwork with hierarchical generator and encoder to generate lyrics given a specified style and topic pair.

1 Introduction

Text generation is one of the most popular task in the natural language processing area. Recently, there are a large amount of works [(Zhang et al., 2017), (Chen et al., 2018), (Ahamad, 2018), (Wang et al., 2018), (Zhang and LeCun, 2018), (Donahue and Rumshisky, 2018)] proposed to generate text using generative adversarial networks (GAN) (Goodfellow et al., 2014). However, most current works only focus on generating one or several sentences, not even a long paragraph, let alone a passage.

Lyrics, as one type of text, has some its own characteristics. Lyrics usually have rhythms, are mostly not very short, and have some repetition patterns. These characteristics make generating lyrics much harder than generating normal texts.

Most current works about lyric generation coming with many conditions, such as given a piece of melody (Watanabe et al., 2018), or only generating a specific type of lyric (Malmi et al., 2016). However, we do not see any prior work about generating lyrics automatically given a style and a topic. Thus we are planning to focus on this novel problem. We are interested to see whether our proposed model can learn different features for different styles, distinguish between various topics, and generate real-looking lyrics.

Our contributions are as follows.

- We build a GAN-like neural network model to generate lyric given a style and a topic.
- We propose a novel hierarchical structure for both lyric generation and encoding.
- We propose two different evaluation methods to quantitatively measure the authenticity of the generated lyric.

2 Related Work

Lyric is a type of text, and we are going to use the idea of GAN to do the generation. In this section, we discuss the related works about text generation using GAN and current works on lyric generation.

Traditional text generation uses variational autoencoder(VAE). Recently more work begins to apply adversarial learning in text generation. Many of them applied a sequence-to-sequence model as a generator and another language model as a discriminator [(Zhang et al., 2017),(Chen et al., 2018)]. Their loss functions are the traditional GAN loss, WGAN loss, or WGAN-GP loss. In 2017, (Yu et al., 2017) first applied reinforcement learning in GAN. They proposed to train the GAN model using policy gradient. Based on their approach, more recent GANs begin to apply reinforcement learning as their loss functions. (Wang et al., 2018) combines the VAE structure with GAN. However, even though there are many trails on adversarial learning in text generation, the performances weren't as satisfactory as their performances in image generation. One big problem is that the generated texts are not grammatically correct. Besides, the syntactic and semantic features are not coherent either. These challenges still need to be solved.

Lyric generation is a specific part of text generation. Many paper focused on generating a lyric for specific

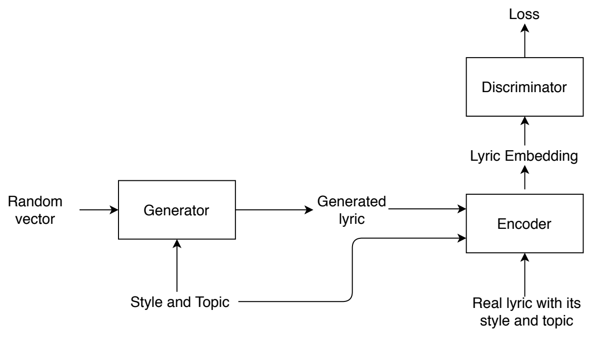

Figure 1: GAN model illustration.

author (Barbieri et al., 2012) or specific style like rap (Malmi et al., 2016). For these specific tasks, they focus on features like rhyme patterns and the connection between contexts. Traditional approach uses hidden markov model (HMM) to represent trasitions betwenn topics (Watanabe et al., 2014). The HMM model can capture the inter-chorus and inter-line relations successfully. More recent methods used RNN to encode the sequences (Watanabe et al., 2018), (Malmi et al., 2016). Another direction of generating the lyrics is generating lyrics based on input melodies(Watanabe et al., 2018). The structure is a language model conditioned on a featurized melody.

3 Task Description

In this work, we focus on generating a song lyric given a style and a topic. Specifying a lyric style, such as rock, country, or jazz, with a content topic, we want to build a GAN-like neural network model to generate a lyric for us automatically. Given the style and topic of a lyric, the generated lyric should look real, have a style same as the given one, and contain the meaning relevant to the topic.

4 Approach

We use a generative adversarial network (GAN) model to do the lyric generation. GAN, by its name, contains two network modules: one generative net and an adversarial net. The generative net tries to generate results that look like real data, while the adversarial net is going to tell whether a data is generated or not. These two components are trained iteratively.

Our model is shown in Fig. 1. It contains three components, a generator, an encoder, and a discriminator. The generator corresponds to the generative net in the GAN framework, while the encoder and discriminator together correspond to the adversarial net. We use the WGAN with gradient penalty (Gulrajani et al., 2017) framework for training. The generator produces lyrics conditioned on the given style and the content topic. The encoder gives the embedding of a lyric. The discriminator is trained to help the generated lyrics look like real ones.

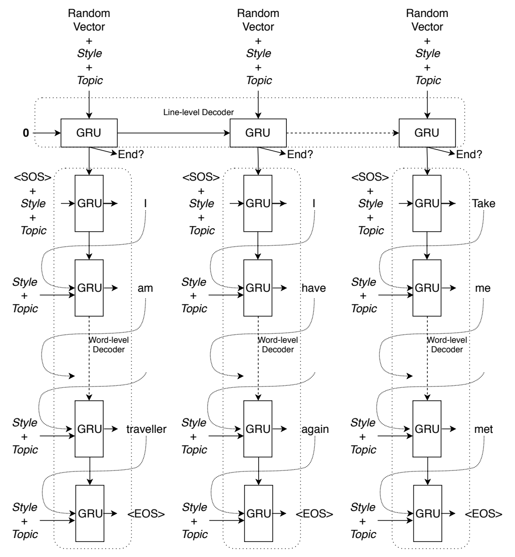

Figure 2: Generator illustration.

4.1 Generator

The structure of the generator is shown in Fig. 2. It is hierarchical, and contains two GRU RNN decoders, a *line-level decoder* and a *word-level decoder*.

The input to the line-level GRU decoder is a normalized random vector drawn from a standard normal distribution, concatenated with the given style and topic. The initial hidden state of line-level GRU is a zero vector. For each step i of the line-level decoder, we try to learn two vectors: one vector e_i indicating whether the current line generated is the last line or not, and the other vector t_i deciding the topic for that line to be generated, which is treated as the initial hidden state of the word-level decoder. The first vector is a binomial distribution over the states $\{$CONTINUE, END$\}$.

The input to the word-level decoder is the word generated in the previous time step concatenated with the given style and topic. For the first time step, we use the start-of-sentence (SOS) tag. For each time step of the word-level decoder, we want it to learn the next word given previous word. We add a linear embedding layer right before the GRU input to reduce the vector dimension. A linear layer with softmax operation is applied on GRU output, which makes the output represent the probability of generating each word over the vocabulary. The end of sentence is marked using an end-of-sentence (EOS) tag.

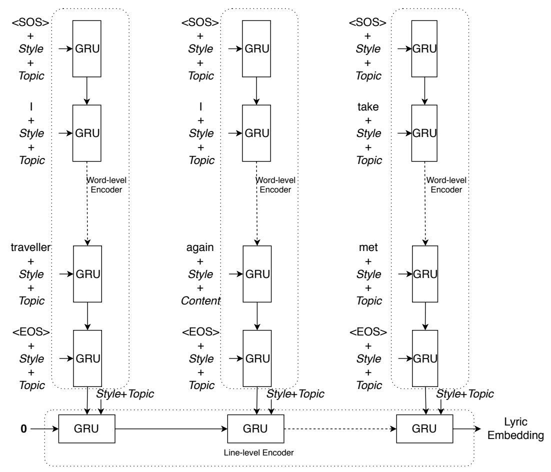

Figure 3: Encoder illustration.

4.2 Encoder

The encoder also has a hierarchical structure, which is illustrated in Fig. 3. Similar to the generator, it contains two GRU RNNs: a *word-level encoder* and a *line-level encoder*. For every word token in a line, we concatenate it with the given style and topic as the input to the word-level encoder. We also add a linear embedding layer right before the GRU input. The final hidden state of the word-level encoder is treated as the embedding of that line. Then we feed the line embedding concatenated with style and topic into the line-level encoder. Then the final hidden state of line-level encoder is the lyric embedding.

Using a hierarchical structure has many advantages. First, it not only considers the relations among words, but also considers the relations among lines. Besides, using a single-level RNN as an encoder or a decoder empirically performs bad for long sequences. In the hierarchical structure, each recurrent unit only processes a small number of tokens, which will make it perform better.

4.3 Discriminator

Like other WGAN-like framworks [(Arjovsky et al., 2017), (Hsu et al., 2017)] the discriminator measures the distance between the generated data distribution and real data distribution. Given a real lyric or a generated lyric, we first use the encoder to obtain the lyric embedding. The lyric embedding is then fed to the discriminator. The discriminator is implemented as a two-layer MLP with leaky ReLU activation.

4.4 Training

To train our model, we employ a two-stage strategy. First we combine the encoder and the generator as an auto-encoder, using the real lyrics to train the auto-encoder. This is going to give the generator and the encoder some language prior. Then we put the pre-trained generator and the encoder, together with the discriminator, into the GAN training procedure.

4.4.1 Pre-training

In the GAN training procedure, the input to the generator is a random vector. Thus without pre-training, it may be hard to generate a well-organized lyric. To avoid this problem, we combine the encoder and the generator as an auto-encoder, and use the real lyrics to train it. We first use the encoder to encode the lyric, and then use the generator to decode the lyric back. The input to the line-level decoder at every time step is the lyric embedding. Here involve two losses, for a lyric with m lines and n_i words on every line. One loss is on the word level, that is, a cross-entropy loss between the generated word $\hat{w_{ij}}$ and the ground-truth word w_{ij}.

$$L_{\mathrm{w}} = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{n_i}\sum_{j=1}^{n_i}\text{cross\_entropy}(\hat{w_{ij}}, w_{ij}) \tag{1}$$

The other loss is on the line level, which measures whether the generator produces the correct number of lines. Cross-entropy loss is applied between the predicted $\{$CONTINUE, END$\}$ $\hat{s_i}$ and the ground-truth state s_i.

$$L_{\mathrm{s}} = \frac{1}{m}\sum_{i=1}^{m}\text{cross\_entropy}(\hat{s_i}, s_i) \tag{2}$$

The final loss for the pre-training is a weighted sum of Eq. 1 and Eq. 2.

$$L_{\mathrm{auto}} = L_{\mathrm{c}} + \lambda_{\mathrm{s}} L_{\mathrm{s}} \tag{3}$$

4.4.2 Adversarial training

After the aforementioned pre-training process, we put our model into the GAN training procedure. Compared to the conventional GAN setting, our generator corresponds to the generative net; while our encoder and discriminator together correspond to the discriminative net. In every iteration of the adversarial training, we first update the parameters of the encoder and the discriminator, then update the parameters of the generator.

We input normalized random noise drawn from a standard normal distribution to the generator. Then we concatenate the output from the softmax layer to get a "generated" lyric, where each token is the probability of words over the dictionary. After we generate a lyric $\hat{z}$, we feed it to the encoder and the discriminator to get a value $D(\hat{z})$.

We also feed a corresponding real lyric z to the encoder and the discriminator to get a value $D(z)$. We apply the WGAN with gradient penalty (WGAN-GP) (Gulrajani et al., 2017) framework to update the network modules. The reason why we use WGAN-GP rather than original GAN is that WGAN-GP framework is empirically more stable and easier to train.

Given fixed generator, the loss to train the encoder and discriminator is as follows, where $E(\cdot)$ is taking the expectation and α and λ_{gp} are tune-able hyper-parameters.

$$\mathrm{GP} = E[(|\nabla D(\alpha z - (1-\alpha \hat{z}))| - 1)^2] \tag{4}$$

$$L_{\mathrm{wd}} = -E[D(z)] + E[D(\hat{z})] \tag{5}$$

$$L_{\mathrm{d}} = L_{\mathrm{wd}} + \lambda_{gp}\mathrm{GP} \tag{6}$$

Given fixed encoder and discriminator, the loss to train the generator is as follows, where $E(\cdot)$ is taking the expectation.

$$L_{\mathrm{g}} = -E[D(\hat{z})] \tag{7}$$

5 Dataset

The dataset we use is the 380000-lyrics-from-metrolyrics dataset available on Kaggle[1]. This dataset contains around 380,000 lyrics with many different topics and styles. The styles of lyrics covered by this dataset are pop, hip-hop, rock, metal, country, jazz, electronic, folk, R&B, indie, and other.

5.1 Data Cleaning

In this work, we only use English lyrics with styles including hip-hop, metal, and country. These three styles of lyric are quite distinguishable and only using English lyrics constrains our vocabulary within English words. We remove all words in brackets or parenthesises and all punctuations. Besides, we replace some abbreviation. For example, we change “don’t” to “do not”. We build a dictionary of 9746 word tokens, including “start-of-sentence” (SOS), “end-of-sentence” (EOS, and “unknown-word” (UNK) tags. For the words not in the dictionary, we replace them with UNK tags. We remove lyrics where more than 50% of the word tokens are UNK tags. We also remove lyrics which is less than 4 lines, and keep the maximum number of words in each line as 32.

After the above data cleaning procedure, we obtain a filtered lyric dataset. The data distribution of the filtered dataset is shown in Tab. 1. We randomly split the filtered dataset into training, validation, and testing splits with a proportion of 8:1:1.

[1]`https://www.kaggle.com/gyani95/380000-lyrics-from-metrolyrics`

Genre	Number of Instances
Hip-Hop	12220
Metal	13776
Country	13748

Table 1: Data Distribution of the Filtered Dataset

5.2 Test Sets

We built three test sets to cover three test cases. Each contains 150 style and topic pairs, 50 for every style (i.e., hip-hop, metal, and country).

Test Set 1. This set contains the style and topic pairs which have been seen during training. This set is randomly selected form the training split.

Test Set 2. This set contains the topics that have not been seen during training and validation. This set is a subset of the testing split.

Test Set 3. This set contains the topics that have been seen during training, but the style and topic pairs have not been seen when training and validation. This set is also a subset of the testing split.

6 Experiments

6.1 Models Compared

We compare the lyrics generated from 4 different models with real lyrics. We refer these models as follows.

Random Weights. This model is the generator with randomly initialized weights.

GAN Scratch. This model is trained using the GAN training procedure (Sec. 4.4.2) from scratch (with randomly initialized weights).

Pre-train Only. This is the generator with the weights pretrained on the auto-encoder task (Sec. 4.4.1).

GAN Pre-train. This model is trained using the GAN training procedure (Sec. 4.4.2) from the pretrained model (Pre-train Only).

6.2 Evaluation Methods

We propose two different evaluation metrics to quantitatively measure our lyric generation models, that is, a discriminator approach and a classification measurement.

Discriminator Approach. The discriminator output (a.k.a. *discriminator score*), $-L_{wd}$ from Eq. 5, measures the distance between the generated data distribution and the real data distribution. We use this discriminator score to measure the “realness” of our generated lyric with our trained discriminator. A real lyric should have a discriminator score of 0, and the smaller the discriminator score, the more real of the generated lyric.

Classification Measurement. We trained another classification network to classify the style of the generated lyric. The classifier is trained on the training data, i.e., real lyrics. This measurement is to see whether our generated lyric actually has the given style. We report the *classification accuracy* for the measurement.

6.3 Implementation Details

We use a single-direction single-layer GRU RNN to implement the line-level decoder, word-level decoder, word-level encoder, and line-level encoder. We set the hidden state dimension to 128 for all of them, and set the embedding vector dimension to 128 for the word-level decoder and word-level encoder.

We feed the output of the line-level decoder to two different network structures. One is a linear layer to produce the binomial distribution over the {CONTINUE, END} states; the other is a two-layer MLP with ReLU activation to produce the initial hidden state for the word-level decoding.

We use one-hot encoding to represent words; thus word encoding has dimension 9746. For topic encoding, we use the pre-trained 300-dimension word2vec (Mikolov et al., 2013) vectors to represent every word in the topic, and then calculate the mean vector over the words to represent the topic, which has dimension 300. We also use one-hot encoding to represent the lyric style.

When training the auto-encoder, we use learning rate 0.0001 with the Adam optimizer (Kingma and Ba, 2014) and a batch size of 30. We set the λ_s in Eq. 3 to 0.1 and train the auto-encoder for 20 epochs.

For the GAN training, we use a batch size of 25 with the Adam optimizer (Kingma and Ba, 2014). The learning rate used for all the network components in the GAN Scratch model is 0.0001. For the GAN Pre-train model, we set the learning rate to 0.000095 for the pretrained generator and encoder and 0.0001 for the discriminator. We set the α in Eq. 4 to 1 and the λ_{gp} in Eq. 6 to 10.

We also trained a classification network to classify the style of a generated lyric. The classification network is a combination of the encoder (without style and topic conditions) and a two-layer MLP with ReLU activation. For training, we use the Adam optimizer (Kingma and Ba, 2014) with learning rate 0.0001 and batch size of 150.

For all the training, we pick the number of epochs used for testing based on validation set performance.

6.4 Quantitive Results

The discriminator score is shown in Tab. 2. We can see that the Pre-trained Only model achieves the best performance in terms of the discriminator score. However, we can not handle the training for the GAN Pre-train model well.

The classification accuracy is displayed in Tab. 3. It is quite sad that all the 4 models do not perform well on this metric.

Model	Test Set 1	Test Set 2	Test Set 3
Random Weights	5.30	5.27	5.31
GAN Scratch	5.15	5.10	5.09
Pre-train Only	1.95	2.08	1.72
GAN Pre-train	5.31	5.31	5.30
Real Lyric	0	0	0

Table 2: Discriminator score (the lower the better).

Model	Test Set 1	Test Set 2	Test Set 3
Random Weights	0.33	0.33	0.33
GAN Scratch	0.34	0.33	0.33
Pre-train Only	0.33	0.33	0.33
GAN Pre-train	0.35	0.35	0.35
Real Lyric	0.86	0.81	0.67

Table 3: Classification accuracy (the higher the better).

6.5 Qualitative Results

Tab. 4 and Tab. 5 show the lyrics generated by the Pre-trained Only model for the topics “Voyage” and “My Love” respectively with different styles. The topic “Voyage” does not exist in the training data while “My Love” exists. We can see that the model can generate lyrics related to the given topic (in terms of some words) and can generate different lyrics for different styles.

7 Discussion

7.1 Lessons Learned and Project Evaluation

In this project, we implemented the hierarchical structure and the adversarial learning framework all by ourselves. The training is hard due to the complicated framework and the large dataset, it takes us around 3 hours to train one epoch for the GAN training process. Our models are not well-trained, especially for the GAN Pre-train model.

Another lesson we learned is the difficulty of text generation. Unlike image generation, text generation must follow grammar rules and must contain coherent semantics. These two characteristics make text generation harder. In our project, we try to distinguish different styles just based on the lyrics. However, even for human beings, it is difficult to distinguish the styles by only looking at the lyrics. In this way, we failed to generate the lyric with the specified style. More sophisticated methods should be applied to solve this problem.

As for the evaluation of our project, we believe that our project is partially successful. On the one hand, we can see that the hierarchical structure works in generating lyrics. In the qualitative results, we can find out that there are somewhat “reasonable” generated lyrics. Besides, the GAN training procedure also increased the style classification accuracy a bit even though it is only trained for 10 epochs.

Style	Voyage
Hip-Hop	i finally know without anything if your low brand song
	i am bombs alone through the brain
	make me joe
	chain in your ears
	let ya
	i can feel you fine songs bread UNK cream liar split michael lonely while they time
	dipping up childhood if u share UNK me
	but it is above it
	see you
	dash girls tummy ella bronx ol audio game floors lace storm awfully sprayed tracks undone
Metal	years like your plague high rocks fallen onward oliver
	slowly people surely raw legs dream freaks praising UNK
	high sun UNK understands you do not enough
	hey UNK
	looking intended top cactus
	let me make UNK dream
	but no one those battlefield
	i want flyer man we hate worm string close UNK itself repeat chains
	baby save your rest UNK me
	man phantom holes
Country	take pride carnage gifts alabama roads end
	meant bible shackles trip sucka direct trophies
	milli tricks around packing nasty track unleash passionate ceremony ocean ei paws
	daydream in season
	hitch up you gave my frosty coast UNK shy correct this lies jose
	on especially tonight
	spider weights without april

Table 4: Qualitative results for the Pre-trained Only model with topic “Voyage”.

Style	My Love
Hip-Hop	gentle winter colder angel confusion turning shitty floor
	seams feeling time always triste finish degree than love
	i drive up UNK tattoos lifetime sho cold pole baby when dreaming waiting time that had time
	here rocking red long
	oh heh
Metal	kiss my bitch that she has up guns children stepping on myself alone
	i are asking it do not love you
	sunny baby up miracle hand do not
	some diamonds do not know it
	my sweet time UNK me
	verse flirting da checks insane knife conceive i laugh words
	holds till least presence time so testify obey friend fist
Country	baby if i can not reasons betrayed away easy awhile autumn torch
	cling UNK biscuits
	rabbit UNK
	yea girl baby is ready when can be myself cocky UNK man
	leaving me buddy
	why do not wap dizzy mall queen nickels game minute brains confusing top tonight

Table 5: Qualitative results for the Pre-trained Only model with topic “My Love”.

On the other hand, according to Tab. 2, the GAN Pretrain model, which is supposed to be the best model, achieves similar scores with the Random Weights model. It is hard to find a good set of hyper-parameters for this model.

7.2 Future Work

For future work, we can try tuning the hyper-parameters of the model to improve the result. From our experiments, we find that it is easy for the discriminator to distinguish generated lyrics from real ones. We can train the generator for more iterations after update the weights of the encoder and the discriminator one time. Besides, more time is needed for training the models.

Text generation is still a challenge task in the natural language processing area. There are other adversarial learning methods using reinforcement learning, such as SeqGAN (Yu et al., 2017) and LeakGAN (Guo et al., 2017), to tackle this task. New structures need to be proposed and improved for the text generation task.

Appendix

The link to the source code is `https://github.com/peterfengyx/Lyric-Generation`; and the link to the original corpus is `https://www.kaggle.com/gyani95/380000-lyrics-from-metrolyrics`.

References

Afroz Ahamad. 2018. Generating text through adversarial training using skip-thought vectors. *arXiv preprint arXiv:1808.08703*.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein gan. *arXiv preprint arXiv:1701.07875*.

Gabriele Barbieri, François Pachet, Pierre Roy, and Mirko Degli Esposti. 2012. Markov constraints for generating lyrics with style. In *Ecai*, volume 242, pages 115–120.

Liqun Chen, Shuyang Dai, Chenyang Tao, Haichao Zhang, Zhe Gan, Dinghan Shen, Yizhe Zhang, Guoyin Wang, Ruiyi Zhang, and Lawrence Carin. 2018. Adversarial text generation via feature-mover's distance. In *Advances in Neural Information Processing Systems*, pages 4671–4682.

David Donahue and Anna Rumshisky. 2018. Adversarial text generation without reinforcement learning. *arXiv preprint arXiv:1810.06640*.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. 2017. Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, pages 5767–5777.

Jiaxian Guo, Sidi Lu, Han Cai, Weinan Zhang, Yong Yu, and Jun Wang. 2017. Long text generation via adversarial training with leaked information.

Chin-Cheng Hsu, Hsin-Te Hwang, Yi-Chiao Wu, Yu Tsao, and Hsin-Min Wang. 2017. Voice conversion from unaligned corpora using variational autoencoding wasserstein generative adversarial networks. *arXiv preprint arXiv:1704.00849*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Eric Malmi, Pyry Takala, Hannu Toivonen, Tapani Raiko, and Aristides Gionis. 2016. Dopelearning: A computational approach to rap lyrics generation. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 195–204. ACM.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Heng Wang, Zengchang Qin, and Tao Wan. 2018. Text generation based on generative adversarial nets with latent variables. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 92–103. Springer.

Kento Watanabe, Yuichiroh Matsubayashi, Kentaro Inui, and Masataka Goto. 2014. Modeling structural topic transitions for automatic lyrics generation. In *Proceedings of the 28th Pacific Asia conference on language, information and computing*.

Kento Watanabe, Yuichiroh Matsubayashi, Satoru Fukayama, Masataka Goto, Kentaro Inui, and Tomoyasu Nakano. 2018. A melody-conditioned lyrics language model. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, volume 1, pages 163–172.

Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2017. Seqgan: Sequence generative adversarial nets with policy gradient. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Xiang Zhang and Yann LeCun. 2018. Adversarially-trained normalized noisy-feature auto-encoder for text generation. *arXiv preprint arXiv:1811.04201*.

Yizhe Zhang, Zhe Gan, Kai Fan, Zhi Chen, Ricardo Henao, Dinghan Shen, and Lawrence Carin. 2017. Adversarial feature matching for text generation. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 4006–4015. JMLR. org.