



Melody Generation from Lyrics using Three Branch Conditional LSTM-GAN

By

Abhishek Srivastava

Under the supervision of

Dr. Rajiv Ratn Shah, IIIT Delhi

Dr. Yi Yu, NII Japan

Indraprastha Institute of Information Technology Delhi

July, 2020

©Indraprastha Institute of Information Technology Delhi
(IIITD), New Delhi, 2020



Melody Generation from Lyrics using Three Branch Conditional LSTM-GAN

By

Abhishek Srivastava

Submitted

in partial fulfillment of the requirements for the degree of
Master of Technology

to

Indraprastha Institute of Information Technology Delhi

July, 2020

Certificate

This is to certify that the thesis titled “**Melody Generation from Lyrics using Three Branch Conditional LSTM-GAN**” being submitted by **Abhishek Srivastava** to Indraprastha Institute of Information Technology Delhi, for the award of the Master of Technology, is an original research work carried out by him under my supervision. In my opinion, the thesis has reached the standards fulfilling the requirements of the regulations relating to the degree.

The results contained in this thesis have not been submitted in part or full to any other university or institute for the award of any degree/diploma.

July 2020

Dr. Rajiv Ratn Shah
Department of Computer Science
Indraprastha Institute of Information Technology Delhi
New Delhi, India

Dr. Yi Yu
Digital Content and Media Sciences Research Division
National Institute of Informatics
Tokyo, Japan

Acknowledgements

I want to thank my thesis advisors Dr. Rajiv Ratn Shah, IIIT Delhi, and Dr. Yi Yu, NII Japan, for their constant support and supervision. I am immensely grateful to them for providing me with their valuable guidance and insights through the course of the thesis. I also want to thank my colleagues at the MIDAS Lab, IIIT Delhi, and Dr. Yu's Lab, NII Japan, who volunteered for the subjective evaluation for this research project. Finally, I want to express my gratitude to my family for providing me with unfailing support and continuous encouragement through researching and writing this thesis.

July 2020

Abhishek Srivastava
Department of Computer Science
Indraprastha Institute of Information Technology Delhi
New Delhi, India

Abstract

Automating the process of melody generation from lyrics has been a challenging research task in the field of artificial intelligence. Lately, however, music-related datasets have become available at large-scale, and with the advancements of deep learning techniques, it has become possible to better explore this task. In particular, Generative Adversarial Networks (GANs) have shown a lot of potential in generation tasks involving continuous-valued data such as images. In this work, however, we explore Conditional Generative Adversarial Networks (CGANs) for discrete-valued sequence generation, in particular, we exploit the Gumbel-Softmax relaxation technique to train GANs for discrete sequence generation. We propose a novel architecture, Three Branch Conditional (TBC) LSTM-GAN for melody generation from lyrics. Through extensive experimentation, we show that our proposed model outperforms the baseline models by generating tuneful and plausible melodies from the given lyrics.

Contents

Certificate	i
Acknowledgements	ii
Abstract	iii
List of Figures	vi
List of Tables	vii
1 Introduction	1
1.1 Motivation and goal of the project	1
1.2 Solution and Challenges	2
2 Background work	3
2.1 Related Words	3
2.1.1 Music Generation	3
2.1.2 Discrete Data Generation using GANs	3
2.2 Basics of music theory	5
2.3 Long-Short Term Memory	7
2.4 Skip-gram models	8
2.5 Generative Adversarial Networks (GANs)	10
2.6 Gumbel-Softmax Relaxtion	12
2.7 Kullback–Leibler divergence	14
2.8 Evaluation Metrics	15

2.8.1	Self-BLEU	15
2.8.2	Maximum Mean Discrepancy (MMD)	16
2.8.3	Subjective Evaluation	16
3	Methodology	18
3.1	Data	18
3.2	Architecture	19
3.2.1	Three Branch Conditional (TBC) LSTM-GAN	20
3.2.2	Single Branch Conditional (SBC) LSTM-GAN	24
4	Experiments and discussion	29
4.1	Experimental Setup	30
4.2	Music Quantitative Evaluation	30
4.3	Quality and Diversity Evaluation	32
4.3.1	Quality Evaluation	32
4.3.2	Diversity Evaluation	33
4.3.3	Subjective Evaluation	34
4.4	Generated music	34
5	Conclusion and future work	36
	Bibliography	41

List of Figures

2.1	Alignment between the lyrics and melody	5
2.2	Relationship between note duration and note	5
2.3	Relationship between rest values and corresponding symbols	5
2.4	Architecture of LSTM cell	7
2.5	Skip-gram model	9
2.6	Generative adversarial network	11
2.7	Conditional adversarial network	12
3.1	Dataset distribution of music attributes	18
3.2	Overall architecture of TBC-LSTM-GAN	21
3.3	Architecture of TBC-LSTM-GAN generator sub-network	22
3.4	Architecture of TBC-LSTM-GAN discriminator	24
3.5	Overall architecture of SBC-LSTM-GAN	25
3.6	Architecture of SBC-LSTM-GAN generator	26
3.7	Architecture of SBC-LSTM-GAN discriminator	27
4.1	Distribution of transitions	32
4.2	Training curves of of MMD scores on testing dataset	33
4.3	Training curves of self-BLEU scores on testing dataset	33
4.4	Subjective Evaluation Results	34
4.5	Sheet music for generated melodies	35

List of Tables

2.1	Correspondance between Note-Pitch	6
4.1	Configuration of the generator and discriminator in TBC-LSTM-GAN	30
4.2	Configuration of the generator and discriminator in SBC-LSTM-GAN	30
4.3	Metrics evaluation of in-songs attributes	31

Chapter 1

Introduction

1.1 Motivation and goal of the project

Songwriting is one of the creative human endeavors. Typically, during songwriting, a musician either writes the lyrics for a song first and then tries to compose a melody to match the lyrics or composes a melody for a song first and then write lyrics to fit the melody. From the perspective of automation of the songwriting process, this gives rise to two very interesting and related problems. These are lyrics generation from a melody and melody generation from lyrics. In this work, we restrict ourselves to the latter one, i.e., melody generation from lyrics.

We aim to build a model for lyrics-conditioned melody generation that can capture the correlations that exist between segments of lyrics and patterns in a melody. We define lyrics as a sequence of syllables and a melody as a sequence of triplets of discrete-valued music attributes. Each triplet of music attributes in a melody is composed of MIDI number, duration, and rest. The MIDI number (or pitch) refers to the frequency of a musical note, the duration refers to the length of time we play a musical note and the rest indicates the length of silence after a playing a musical note. The data we need for this task is composed of a collection of melody-lyrics pairs where the melody and lyrics are aligned note-syllable level i.e., corresponding to each triplet of discrete-valued music attributes in a melody we have a syllable in the lyrics.

1.2 Solution and Challenges

In a general sense, we can consider the task of melody generation from lyrics as a sequence-to-sequence task whereby we are given a lyrics i.e. a sequence of syllables and we want to generate a melody i.e., a sequence of discrete-valued triplets of music attributes. In this work, we explore the framework of Generative Adversarial Networks (GANs) as a potential solution for the task at hand. More precisely, we propose a novel architecture, Three Branch Conditional (TBC) LSTM-GAN, which generates melody conditioned on the lyrics.

The problem of melody generation from lyrics is intrinsically challenging in nature due to the existence of a many-to-many mapping between a melody and lyrics. It means that for a given melody there can exist multiple lyrics and for a given lyrics there can exist multiple melodies i.e. there is no fixed melody for the given lyrics.

This poses a real challenge, especially, during the evaluation phase, because, as such, there does not exist any single metric that can measure the generated melody for the given lyrics both qualitatively and quantitatively. Moreover, measuring the quality of a melody is subjective in nature. Another important issue that we encounter is the inability to train GANs for discrete data generation. As we will see later, we employ the Gumbel-Softmax relaxation technique to overcome this issue.

First, some useful research background is presented in Chapter 2 and the architecture of the used model is detailed in Chapter 3. Experiments are presented in Chapter 4. Conclusion and future works are discussed in Chapter 5.

Chapter 2

Background work

2.1 Related Words

2.1.1 Music Generation

Automating the process of music composition through computational techniques has been studied since the 1950s [Hiller and Isaacson, 1958]. More recently, [Rodriguez and Vico, 2014] conducted an elaborate study related to music composition through algorithmic methods. Lately, various techniques of shallow and deep learning have shown promising results for lyrics assisted melody generation. [Ackerman and Loker, 2016] proposed a songwriting system, ALYSIA, based on Random Forest that generates a set of sample melodies when given a short lyrical phrase. [Bao et al., 2018] adopted a sequence-to-sequence framework to develop a melody composition model. To produce notes together with the corresponding alignment they use a hierarchical decoder and a couple of encoders for the lyrics and context melody. [Yu and Canales, 2019] have proposed an LSTM based GAN architecture that uses a quantization framework to discretize its continuous-valued output and generate melodies for the given lyrics.

2.1.2 Discrete Data Generation using GANs

The task of melody generation from lyrics is one of conditional discrete-sequence generation. In this work, we intend to harness the power of GANs as a potential

solution to our problem. GANs were proposed by [Goodfellow et al., 2014], and designed originally to generate continuous-valued data like images. Lately, however, GANs have been used for sequence generation tasks, specifically generation of text, which is discrete in nature. However, using GANs for discrete sequence generation is not straightforward and an active area of research. The reason being, during discrete data generation, the output of the generator is non-differentiable due to which standard gradient-based algorithms cannot be used to train the model. Recently, two types of strategies have been explored to deal with the non-differentiability problem arising from discrete data generation using GANs. The first one is based on the REINFORCE algorithm [Williams, 1992] and the second one is based on reformulating the problem in the continuous space.

More recently, a wide variety of GANs for discrete data generation, especially for text generation, have relied on the RL methods to train the model. SeqGAN [Yu et al., 2017] for instance, bypasses the generator differentiation problem through the use of policy gradient methods [Sutton et al., 2000]. MaliGAN [Che et al., 2017] overcomes the difficulty of backpropagation through discrete random variables by optimizing a low variance maximum-likelihood objective. RankGAN [Lin et al., 2017] uses a rank-based discriminator instead of the binary classifier and is optimized through policy gradient techniques. LeakGAN [Guo et al., 2018] lets the discriminator leak high-level features it has learned to the generator and guides it. In this way, it allows the generator to incorporate intermediate information during each generation step. MaskGAN [Fedus et al., 2018] resorts to a seq2seq model and propose an action-critic CGAN that fills the missing text missing by conditioning on the context surrounding it.

Lately, GANs, for discrete data generation, have relied on either working in a continuous space or approximating the discreteness. Conditional LSTM-GAN [Yu and Canales, 2019] for lyrics-assisted melody generation, for instance, uses a quantization framework to discretize the continuous-valued output it generates. The quantization scheme they adopt constraints the generated continuous-valued music attributes to their closest discrete value. TextGAN [Zhang et al., 2017] and FM-GAN [Chen et al., 2018] apply an annealed softmax relaxation to approximate the

argmax operation at the output of the generator. ARAE [Zhao et al., 2018] uses an auto-encoder module to transform discrete data into a continuous latent space to avoid the issue of non-differentiability. RELGAN [Nie et al., 2019] applies the gumbel-softmax relaxation for training GANs on discrete data.

2.2 Basics of music theory

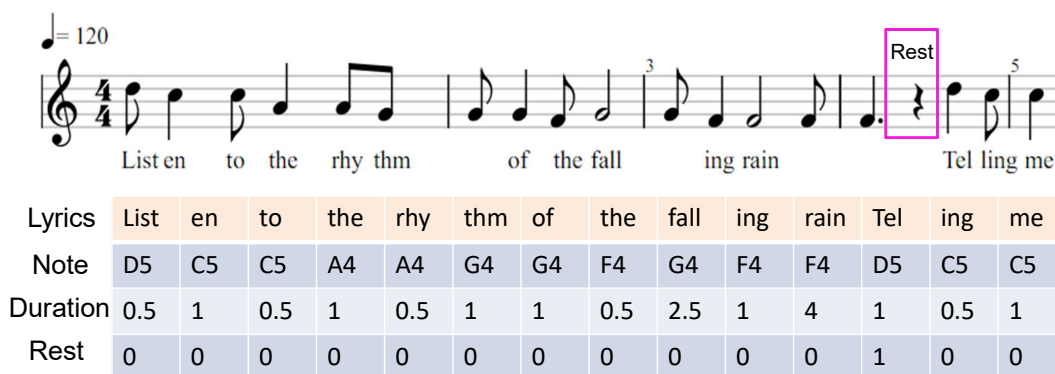


Figure 2.1: Alignment between the lyrics and melody

The data for our task is composed of a collection of melody-lyrics pairs aligned at the note-syllable level. In the music terminology, each such melody-lyrics pair in the dataset is known as the digital music score. Formally, a music score is composed of a melody score augmented with syllable information for each melody note. The digital music score is helpful in analyzing the correlations that exist between the melody and its corresponding lyrics.

0.25	0.5	0.75	1	1.5	2	3	4	6	8	16	32

Figure 2.2: Relationship between note duration and note

0	1	2	4	8	16	32
No rest						

Figure 2.3: Relationship between rest values and corresponding symbols

As we observe from the figure 2.1, we play a single note at a time, therefore we can say that a melody is monophonic in nature. The left-most symbol in a music score is known as the *G-clef* and it denotes the reference pitch. The vertical position of a music note relative to the reference pitch decides the pitch of a note. In a music score, a *measure* is composed of 4 beats and is denoted by a vertical line. The duration of a musical note and the length the rest that follows a note depends on the symbol used to depict it. In figure 2.2 we show the relationship between a musical note (depicted by a symbol) and the value of duration. In figure 2.3 we show the correspondence between values of rest and symbols used to denote it.

Note	MIDI number	Pitch
B3	59	247 Hz
C4	60	262 Hz
C \sharp 4 or D \flat 4	61	277 Hz
D4	62	294 Hz
D \sharp 4 or E \flat 4	63	311 Hz
E4	64	330 Hz
F4	65	349 Hz
F \sharp 4 or G \flat 4	66	370 Hz
G4	67	392 Hz
G \sharp 4 or A \flat 4	68	415 Hz
A4	69	440 Hz
A \sharp 4 or B \flat 4	70	466 Hz
B4	71	494 Hz
C5	72	523 Hz

Table 2.1: Correspondance between Note-Pitch

For instance, a *quarter note* depicted by ♩ lasts a quarter measure or a single beat. An *eighth note* depicted ♩ lasts an eighth measure or a half-beat. Similarly, a *quarter rest* depicted by ♩ lasts a quarter measure or a single beat. When we place a dot next to a musical note or a rest, then its duration is extended by a factor of 1.5. As an example, ♩ lasts for $1/4 \times 1.5 = 3/8$ measure (or 1.5 beat). We denote a musical note by a letter (*C, D, E, F, G, A* or *B*) – and optionally a symbol \flat (flat) or \sharp (sharp)–, followed by a number (which indicates in which frequency range the

note is played)¹. In Table 2.1², we show the correspondence that exists between the musical note, MIDI number, and the pitch.

2.3 Long-Short Term Memory

Long-Short Term Memory (LSTM) is a variant of Recurrent Neural Networks (RNNs) proposed by [Hochreiter and Schmidhuber, 1997]. Unlike the standard RNN architecture, the design of LSTM enables it to learn long term dependencies that exist in sequential data and overcome the problem of vanishing gradient [Pascanu et al., 2012] suffered by RNNs.

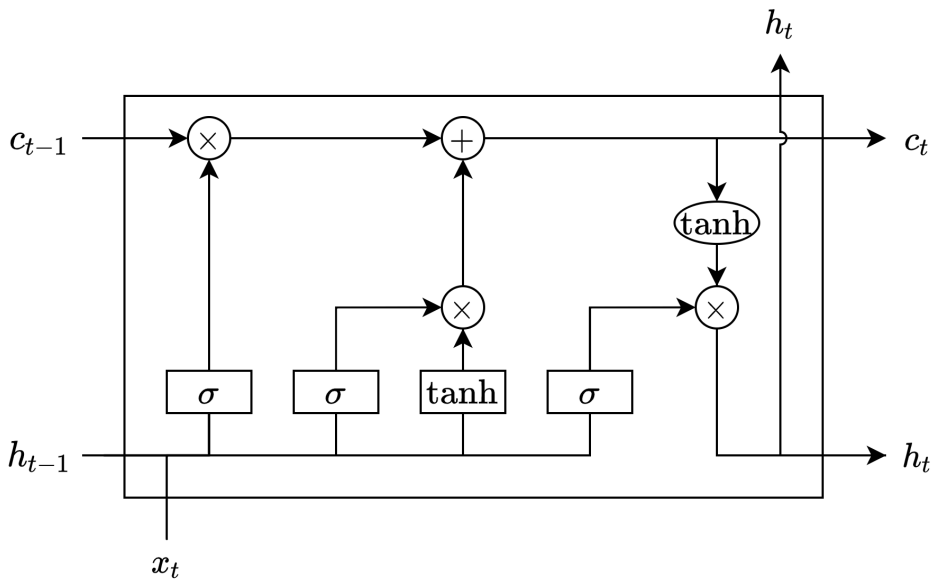


Figure 2.4: Architecture of LSTM cell

Figure 2.4 shows the general architecture of the LSTM cell. LSTM units are composed of three gates namely the input, forget, and the output gate. These three gates are used to control the information flow in the LSTM cell. The key aspect of the LSTM cell is the cell state, c_t . The input gate is used to add new information

¹https://en.wikipedia.org/wiki/Musical_note

²<https://newt.phys.unsw.edu.au/jw/notes.html>

to the cell state, the forget gate is used to forget information from the cell state and the output gate is used to compute the output, by filtering the contents of the cell state. At time-step t , the three states of the gates of the LSTM cells are given by:

$$i_t = \sigma(w_i[h_{t-1}, x_t] + b_i) \quad (2.1)$$

$$f_t = \sigma(w_f[h_{t-1}, x_t] + b_f) \quad (2.2)$$

$$o_t = \sigma(w_o[h_{t-1}, x_t] + b_o) \quad (2.3)$$

where i_t , f_t and o_t denote the input, forget and output gates, h_{t-1} is the output state of the LSTM at previous time-step, w 's and b 's are weights and biases, and x_t is the input of the LSTM cell. Then, the current output of the cell is computed by:

$$h_t = o_t \circ \tanh(c_t) \quad (2.4)$$

where \circ denotes the point-wise multiplication between vectors, and $c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$, with $\tilde{c}_t = \tanh(w_c[h_{t-1}, x_t] + b_c)$.

2.4 Skip-gram models

Skip-gram model with negative sampling famously known as Word2Vec was proposed by [Mikolov et al., 2013]. It is a prediction based model that is used to learn dense representations for words in a corpus. In this work, we use two pre-trained skip-gram models as lyric-encoders. The first skip-gram model is trained on the syllable-level lyrics data to learn syllable-level embeddings. The second skip-gram model is trained on the word-level lyrics data to learn word-level embedding.

To train a Word2Vec model we consider the task of language modelling. The training data can be any text corpus such as Wikipedia. Figure 2.5 shows the architecture of skipgram model. Formally, let D represent the set of correct word pairs (w, c) in the corpus. And, let D' represent the set of incorrect word pairs (w, r) in the corpus. Let v_w , u_c , u_r be the representation of the word w and context word c and r respectively. Considering all $(w, c) \in D$, we want to

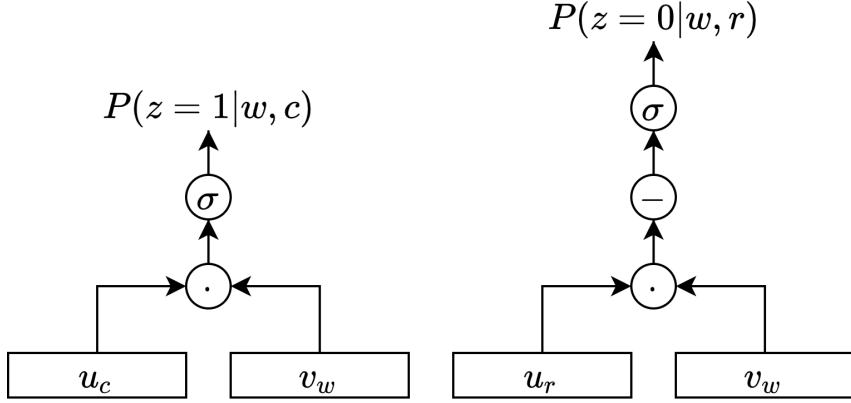


Figure 2.5: Skip-gram model

$$\underset{\theta}{\text{maximize}} \prod_{(w,c) \in D} P(z = 1|w, c) \quad (2.5)$$

where, for a given (w, c) , $P(z = 1|w, c) = \sigma(u_c^T v_w)$ and θ is the word representation (v_w) and context representation (u_c) for all words in our corpus. Considering all $(w, r) \in D'$, we want to

$$\underset{\theta}{\text{maximize}} \prod_{(w,r) \in D'} P(z = 0|w, r) \quad (2.6)$$

where, for a given (w, r) , $P(z = 0|w, r) = 1 - \sigma(u_r^T v_w)$. Combining equations 2.5 and 2.6,

$$\underset{\theta}{\text{maximize}} \prod_{(w,c) \in D} P(z = 1|w, c) \prod_{(w,r) \in D'} P(z = 0|w, r) \quad (2.7)$$

$$\underset{\theta}{\text{maximize}} \prod_{(w,c) \in D} P(z = 1|w, c) \prod_{(w,r) \in D'} (1 - P(z = 1|w, r))$$

$$\underset{\theta}{\text{maximize}} \sum_{(w,c) \in D} \log P(z = 1|w, c) \sum_{(w,r) \in D'} \log(1 - P(z = 1|w, r))$$

$$\underset{\theta}{\text{maximize}} \sum_{(w,c) \in D} \log \sigma(u_c^T v_w) - \sum_{(w,r) \in D'} \log \sigma(-u_r^T v_w)$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$. We sample k negative (w, r) pairs for every positive (w, c) pair hence the size of D' is k times the size of D . In figure 2.5 the skipgram model is shown.

2.5 Generative Adversarial Networks (GANs)

GANs were introduced by [Goodfellow et al., 2014] and since then have been applied successfully to various generation tasks. In this section, we offer a brief overview of GANs. We will also discuss its conditional version, i.e., CGANs introduced by [Mirza and Osindero, 2014].

GANs

A GAN is composed of two models: a generative model G and a discriminative model D . The goal of the generator is to learn the distribution of the training data whereas the goal of the discriminator is to distinguish between the samples from the training data and the generated samples. The training procedure of the generator is designed to maximize the probability of the discriminator making a mistake. Therefore, in a way, the GAN framework corresponds to a minimax two-player game. The architecture of a generator and discriminator can correspond to any non-linear function, such as a multi-layered perceptron (MLP).

The generator learns a mapping, $G(z; \theta_g)$ from a prior noise distribution $p_z(z)$ to the data space to capture the generator distribution p_g over the data x . The discriminator, $D(x; \theta_d)$, outputs a probability estimate that x is sampled from the training data instead of p_g . Figure 2.6 shows the architecture of a GAN.

We train the generator and discriminator simultaneously such that, the parameters of the generator are adjusted to minimize $\log(1 - D(G(z)))$ while parameters of the discriminator are adjusted to minimize $\log(D(X))$. The contradictory objectives of the G and D are captured by the value function $V(G, D)$ given by

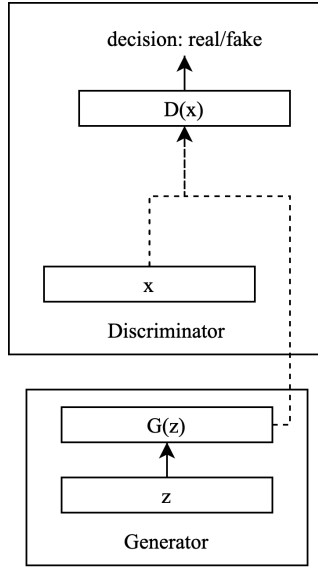


Figure 2.6: Generative adversarial network

$$\min_G \max_D \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [\log(1 - D(G(\mathbf{z})))] \quad (2.8)$$

Conditional GANs

We can easily extend a GAN to a conditioned model by conditioning both the generator and discriminator by some additional information y . Here, y can be any kind of auxiliary information, such as class labels or data from other modalities.

For our problem of lyrics assisted melody generation, as we will see in chapter 3, we condition both the generator and discriminator model by a sequence of syllables i.e. lyrics. Figure 2.7 shows the architecture of CGAN. We combine the prior input noise $p_z(z)$ and y in a joint hidden representation in the generator network. In the discriminator, we present x and y as inputs. The objective function of a two-player minimax game is given as

$$\min_G \max_D \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} [\log D(\mathbf{x}|\mathbf{y})] + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}(\mathbf{z})} [\log(1 - D(G(\mathbf{z}|\mathbf{y})))] \quad (2.9)$$

where \mathbf{y} is the condition vector.

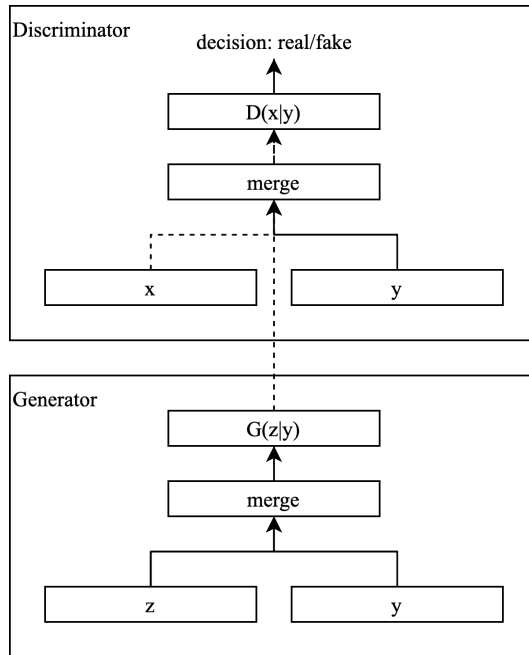


Figure 2.7: Conditional adversarial network

2.6 Gumbel-Softmax Relaxation

Training GANs using discrete-valued data such as text leads to the critical issue of non-differentiability. Due to this, the parameters of the generator cannot be updated and hence the generator cannot learn during the training phase. To mitigate this problem of non-differentiability, and enable the training of the generator various techniques have been used in the past. Mostly these revolve around the REINFORCE algorithm [Williams, 1992] and its variants that have their roots in reinforcement learning (RL).

More recently, the Gumbel-Softmax relaxation technique has been used to train GANs for text generation, a discrete data generation task [Nie et al., 2019]. In contrast to the usage of RL heuristics, using Gumbel-Softmax simplifies our model and allows us to remain in the realm of the GAN framework. Therefore, in this work, we use the Gumbel-Softmax relaxation to enable the training of GANs for discrete data generation. Before we discuss the Gumbel-Softmax relaxation, we consider the task of text generation using a GAN with a recurrent neural network (RNN) based

generator to better understand the issue of non-differentiability.

The task of text generation involves generating a sequence of tokens such that each token belongs to the vocabulary V of the underlying data. When using an RNN based generator, we generate one token at a time step. Let $|V|$ denote the size of the vocabulary and let $o_t \in \mathbb{R}^{|V|}$ denote the output logits, at time t . Then, we can obtain the next generated one-hot token $y_{t+1} \in \mathbb{R}^{|V|}$ by sampling given as

$$y_{t+1} \sim \text{softmax}(o_t) \tag{2.10}$$

where $\text{softmax}(o_t)$ represents a multinomial distribution over the set of all possible tokens present in V . The sampling from a multinomial distribution depicted in equation 2.10 is a non-differentiable operation, this implies the presence of a step function at the output of the generator. Since the derivative of a step function is zero, we have $\frac{\partial y_{t+1}}{\partial \theta_G} = 0$. Using the chain rule, we can see that the gradients of the generator loss l_G w.r.t. θ_G will be

$$\frac{\partial l_G}{\partial \theta_G} = \sum_{i=0}^{T-1} \frac{\partial y_{t+1}}{\partial \theta_G} \frac{\partial l_G}{\partial y_{t+1}} = 0 \tag{2.11}$$

Now, because $\frac{\partial l_G}{\partial \theta_G}$ is zero, an update signal cannot flow back to the generator via the discriminator and hence the generator cannot learn. This is the non-differentiability problem encountered in GANs during discrete data generation.

By applying the Gumbel-Softmax relaxation we can solve the problem of non-differentiability encountered during the training of GANs for discrete data generation. The Gumbel-Softmax approximates samples from a categorical distribution by defining a continuous distribution over a simplex [Jang et al., 2016, Maddison et al., 2016]. The Gumbel-Softmax relaxation is composed of two parts which include the Gumbel-Max trick and relaxation of the discreteness. Using the Gumbel-Max trick we can reparameterize the non-differentiable sampling operation in equation 2.10 as

$$y_{t+1} = \text{one_hot}(\arg \max_{1 \leq i \leq |V|} (o_t^{(i)} + g_t^{(i)})) \tag{2.12}$$

where $o_t^{(i)}$ denotes the i^{th} entry of o_t and $g_t^{(i)}$ is from *i.i.d* standard Gumbel

distribution i.e. $g_t^{(i)} = -\log(-\log U_t^{(i)})$ with $U_t^{(i)} \sim Uniform(0, 1)$. Reparameterization of the sampling operation in equation 2.10 is one part of the solution since the argmax operation in equation 2.12 is non-differentiable in nature. So, now we relax the discreteness by approximating the “one-hot with argmax” by softmax given as

$$\hat{y}_{t+1} = \sigma(\beta(o_t + g_t)) \quad (2.13)$$

where $\beta > 0$ is a tunable parameter called *inverse temperature*. Since \hat{y}_{t+1} in equation 2.13 is differentiable w.r.t. o_t , we can use it instead of y_{t+1} as the input to the discriminator.

2.7 Kullback–Leibler divergence

Kullback–Leibler(KL) divergence is used as a measure to calculate the difference between two probability distributions. It assumes a value of 0 when the two probability distributions are same. Formally, let us consider a random variable $X = \{x_1, \dots, x_n\}$. Let $Y = \{y_1, \dots, y_n\}$ and $\hat{Y} = \{\hat{y}_1, \dots, \hat{y}_n\}$ be the true and predicted probability distribution of X . Then the entropy is given by

$$H_Y = -\sum_{i=1}^n y_i \log y_i \quad (2.14)$$

and cross entropy are given by

$$H_{Y, \hat{Y}} = -\sum_{i=1}^n y_i \log \hat{y}_i \quad (2.15)$$

using 2.14 and 2.15, we can define the KL-divergence by

$$\begin{aligned} D_{KL}(Y \parallel \hat{Y}) &= H_{Y, \hat{Y}} - H_Y \\ &= -\sum_{i=1}^n y_i \log \hat{y}_i + \sum_{i=1}^n y_i \log y_i \end{aligned} \quad (2.16)$$

2.8 Evaluation Metrics

Evaluation of GANs is an active area of research [Theis et al., 2015, Semeniuta et al., 2018]. One of the challenges we face during the evaluation of generated samples is the lack of a single metric that can be used to simultaneously measure the quality and diversity of generated samples. Moreover, as we have mentioned earlier, measuring the quality of the generated melody is subjective in nature and depends on various factors.

Nevertheless, we use the Self-BLEU score [Zhu et al., 2018] to measure the diversity of generated samples and Maximum Mean Discrepancy (MMD) [Gretton et al., 2012] to measure the sample quality. Further, we conduct a subjective evaluation to measure the quality of the sample.

2.8.1 Self-BLEU

The Self-BLEU score is to measure the diversity of the generated samples. To measure the diversity of samples generated by GANs is an important aspect of evaluating GANs since they are susceptible to mode collapse. The value of the Self-BLEU score ranges between 0 and 1. The diversity of the generated samples is inversely proportional to the value of Self-BLEU. The lower the value of Self-BLEU the higher the sample diversity and the lesser the chance of mode collapse of the GAN model. Similarly, the higher the value of Self-BLEU the lower the sample diversity and more the chance of mode collapse of the GAN model.

We know that the BLEU score [Papineni et al., 2002] aims to assess the similarity between two sentences, and therefore, it can be used to evaluate the resemblance between one sentence and rest in a generated collection. By considering one generated sample as a hypothesis and the rest of the generated samples as reference, we can calculate the BLEU score for each generated sample and define the average BLEU score to be the Self-BLEU score.

2.8.2 Maximum Mean Discrepancy (MMD)

The objective of GAN training is that the generator is able to learn the underlying distribution of the training data. Therefore, it is intuitive to assume that if the generator is indeed able to learn the underlying distribution of the data then it will be able to generate high-quality samples. The MMD estimator is helpful to estimate and compare the statistical distributions when given two sets of samples. The value of MMD ranges between 0 and 1 indicating how likely two sets of samples are coming from the same distribution. A value of 0 indicates that the two sets are sampled from the same distribution.

To compare two statistical distributions, the MMD is used. Let \mathcal{F} be a reproducing kernel Hilbert space (RKHS), with the kernel function $k(x, x') := \langle \phi(x), \phi(x') \rangle$, with continuous feature mapping $\phi(x) \in \mathcal{F}$ from each $x \in \mathcal{X}$. Assume $x \sim P_x$ and $y \sim P_y$. Then,

$$\text{MMD}^2(\mathcal{F}, P_x, P_y) = \mathbb{E}[k(x, x')] - 2\mathbb{E}[k(x, y)] + \mathbb{E}[k(y, y')]. \quad (2.17)$$

If P_x and P_y are different, an unbiased quadratic-time estimate of the MMD exists, and the following statements are proven in [Gretton et al., 2012]. Let $X_m := \{x_1, \dots, x_m\}$ and $Y_n := \{y_1, \dots, y_n\}$ be two sets of independently and identically distributed (i.i.d) samples from P_x and P_y respectively. Then, an unbiased empirical estimate of MMD^2 is given by:

$$\begin{aligned} \text{MMD}_u^2(\mathcal{F}, X_m, Y_n) &= \frac{1}{m(m-1)} \sum_{i=1}^m \sum_{j \neq i}^m k(x_i, x_j) \\ &+ \frac{1}{n(n-1)} \sum_{i=1}^n \sum_{j \neq i}^n k(y_i, y_j) - \frac{2}{mn} \sum_{i=1}^m \sum_{j=1}^n k(x_i, y_j). \end{aligned} \quad (2.18)$$

2.8.3 Subjective Evaluation

To measure the quality of a melody generated for a given lyrics is a challenging task. A melody should be plausible as well as tuneful. As mentioned before, there does not exist any metric which can be used to measure the quality of a melody accurately.

This warrants the need for subjective evaluation [Yu and Canales, 2019].

To this end we select 3 different lyrics randomly from the dataset and according we obtain melodies using the baseline model, the proposed model and the ground truth. We use Synthesizer V ³ to synthesize the generated melody and lyrics. We invite [number] random subjects to listen to the generated melodies. We play the lyrics synthesized melody in random order three times. For each melody played, we ask the subjects to answer three questions

1. *How about the entire melody?*
2. *How about the rhythm?*
3. *Does the melody fit the lyrics well?*

The subjects give a score between 1 and 5 for each question. Here, a score of 1 corresponds to very bad, 2 to bad, 3 to OK, 4 to good, and 5 to very good.

³<https://synthesizerv.com/en/>

Chapter 3

Methodology

In this chapter, we will describe the dataset and the architecture of the proposed model, Three Branch Conditional(TBC) LSTM-GAN. Also, we describe the architecture of one of the baseline models, Single Branch Conditional (SBC) LSTM-GAN.

3.1 Data

The data [Yu and Canales, 2019] for our task is composed of a collection of melody-lyrics pairs aligned at the note-syllable level. In the data, a melody is represented by a sequence of triplets of music attributes. Each triplet is composed of the MIDI number of the musical note, the duration of the note, and the length of rest that follows the note. Further, a lyrics is represented by a sequence of syllables.

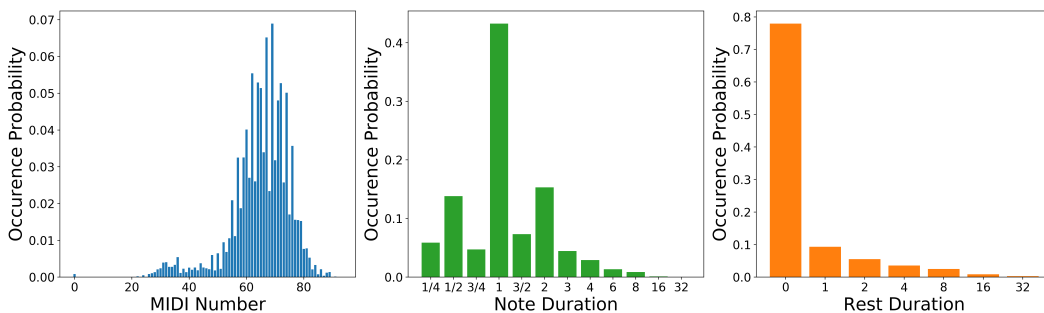


Figure 3.1: Dataset distribution of music attributes

MIDI number is a discrete variable and it can take 128 possible values rang-

ing from 0 to 127. The temporal attributes, duration, and rest are also discrete in nature and assume values that represent the time they last with a single beat represented by a unity. In figure 3.1, we can observe the distribution of MIDI numbers, note duration and rest values in our dataset respectively. In the data, there are 100 distinct MIDI numbers, 12 distinct duration values, and 7 distinct rest values present. In this work, we work with sequences of length 20 i.e. each lyrics contains 20 syllables and each melody contains 20 triplets of music attributes, one triplet of music attribute corresponding to each syllable in the lyrics.

As we have mentioned in section 2.4, in this work, two pre-trained skip-gram models are used as lyric-encoders. To encode the lyrics, we need to get the embedded representation for each syllable in the lyrics. We obtain the embedded representation for a syllable in two steps. In the first step, we use the syllable-level skip-gram to obtain a syllable-level embedding. In the second word, we fetch the representation of the word corresponding to the syllable using the word-level skip-gram model. We then concatenate the syllable-level and word-level representations to obtain the syllable embedding.

Formally, we denote $E_w(\cdot)$ and $E_s(\cdot)$ as word-level and syllable-level skip-gram models respectively. Let s be a syllable from word w present in a lyrics then a dense representation for a syllable is given by $E_w(w)||E_s(s)$ where $E_w(w) \in \mathbb{R}^{10}$ and $E_s(s) \in \mathbb{R}^{10}$ therefore the dimensionality of overall representation for a syllable is 20.

Our goal is to generate a melody, a sequence of triplets of music attributes denoted by $Y = [y_1, \dots, y_T]$ from the given lyrics, a sequence of syllables denoted by $X = [x_1, \dots, x_T]$. Here, $y_i = \{y_i^{MIDI}, y_i^{duration}, y_i^{rest}\}$ represents the i^{th} triplet of music attributes present in the melody, and x_i represents the i^{th} syllable present in the lyrics.

3.2 Architecture

In this section, we describe the architecture of the proposed model for melody generation from lyrics. We train the proposed model by considering the alignment

relationship between the melody and lyrics. The proposed model is composed of three main components: a LSTM-based generator to model long term dependencies, the Gumbel-Softmax relaxation to facilitate the training of the GAN on discrete data and a LSTM-based discriminator for long-distance dependency modeling and to provide meaningful updates to the generator.

We know that a melody is a sequence of “triplets of music attributes” such that each triplet is composed of a MIDI number, duration of musical note and a rest value. There are two key strategies we can adopt to design the generator for melody generation conditioned on the lyrics. In the first strategy, we build a generator to output three sequences : a sequence of MIDI numbers, a sequence of duration values and a sequence rest values. We can then align these three sequences to obtain the melody. In the second strategy, we build a generator to output a single sequence of triplets of music attributes. In this work, we will use both the strategies and show empirically, Three Branch Conditional (TBC) LSTM-GAN designed using the first strategy is better than Single Branch Conditional (SBC) LSTM-GAN designed using the second strategy.

3.2.1 Three Branch Conditional (TBC) LSTM-GAN

Figure 3.2 shows the overall architecture of the TBC LSTM-GAN model.

Generator

The role of generator is to generate a melody conditioned on the lyrics. The generator is composed of three identical and independent sub-networks responsible for generating a sequence of MIDI numbers, a sequence duration values and a sequence of rest values respectively. We align the three output sequences to assemble a melody for the given lyrics.

Formally, we are given a lyrics i.e. a sequence of syllables $X = [x_1, \dots, x_T]$ and we want to generate a sequence of MIDI numbers $Y^{MIDI} = [y_1^{MIDI}, \dots, y_T^{MIDI}]$, a sequence of duration values $Y^{duration} = [y_1^{duration}, \dots, y_T^{duration}]$ and a sequence of rest values $Y^{rest} = [y_1^{rest}, \dots, y_T^{rest}]$. As mentioned earlier, in this work, we use sequences of length $T = 20$. We know, there are 100 distinct MIDI numbers, 12

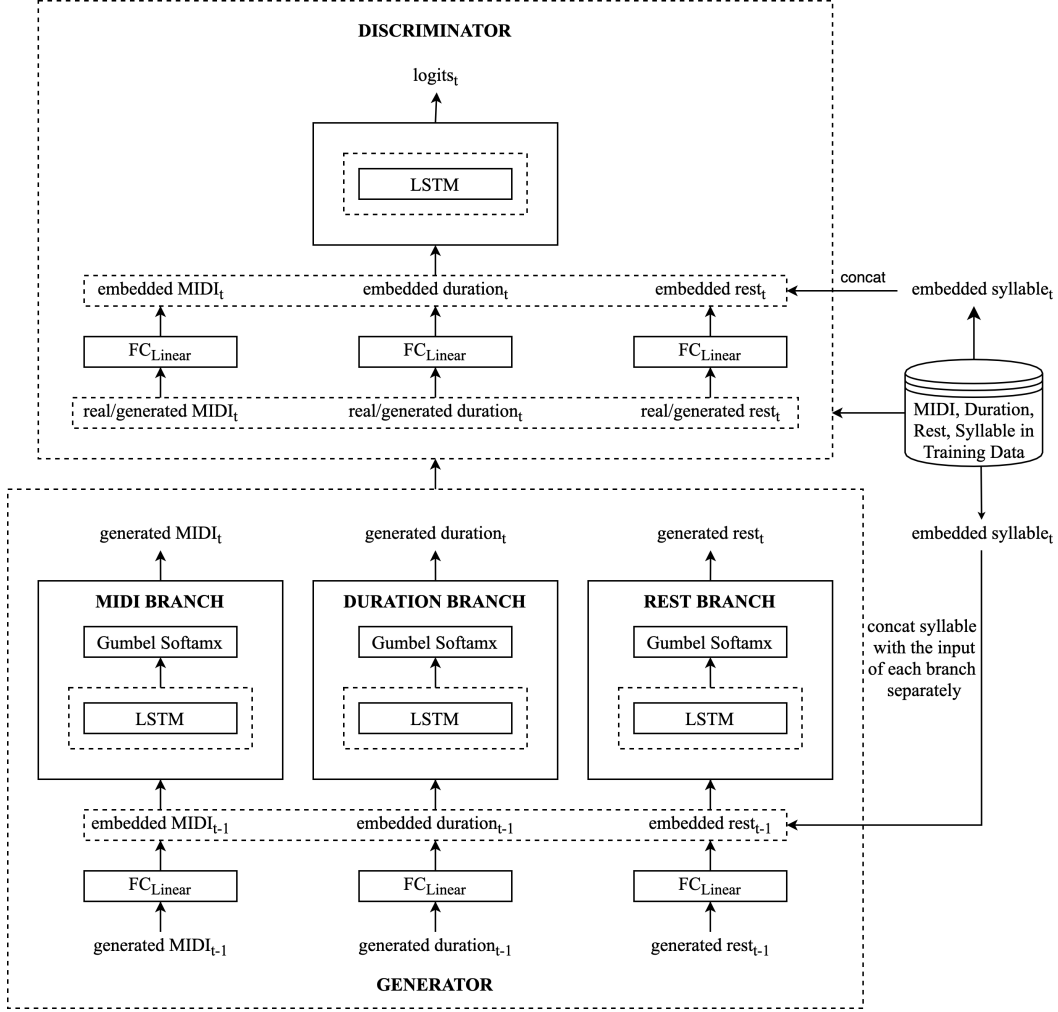


Figure 3.2: Overall architecture of TBC-LSTM-GAN

distinct duration values and 7 distinct rest values present in the data therefore we can represent a MIDI number, duration value and rest value by a 100, 12, 7 dimensional one-hot vector respectively.

Since each sub-network in the generator is identical in nature, we will only explain the forward pass of the sub-network responsible for generating a sequence of MIDI numbers. In Algorithm 1 we provide a succinct description of the forward pass and in figure 3.3, we show the architecture of the sub-network. Formally, we are given a lyrics i.e. a sequence of syllables $X = [x_1, \dots, x_T]$ such that $x_i \in \mathbb{R}^{20}$ and we want to generate a sequence of MIDI numbers $\hat{Y}^{MIDI} = [\hat{y}_1^{MIDI}, \dots, \hat{y}_T^{MIDI}]$ such that

$$\hat{y}_i^{MIDI} \in \mathbb{R}^{100}.$$

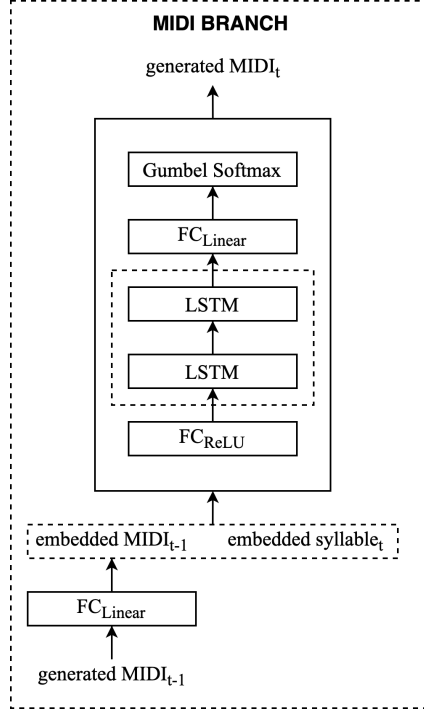


Figure 3.3: Architecture of TBC-LSTM-GAN generator sub-network

Algorithm 1: TBC-LSTM GAN generator sub-network forward pass

- 1 Sample \hat{y}_0^{MIDI} from $Uniform(0, 1)$
 - 2 **for** $t \leftarrow 1, \dots, T$ **do**
 - 3 $z_t \leftarrow FC_{relu}(FC_{linear}(\hat{y}_{t-1}^{MIDI}) || x_t)$
 - 4 $z_t \leftarrow LSTM(z_t)$
 - 5 $\hat{y}_t^{MIDI} \leftarrow GumbelSoftmax(FC_{linear}(z_t))$
 - 6 **end**
-

At the t^{th} timestep, the input to the sub-network is the syllable x_t and the MIDI number generated during the previous timestep \hat{y}_{t-1}^{MIDI} . To obtain an embedding or dense representation of the \hat{y}_{t-1}^{MIDI} we pass it through a fully connected layer with a linear activation. We learn the embedding of MIDI number during the training. We then concatenate the syllable x_t with the embedded MIDI representation and pass it through a fully connected layer with ReLU activation to obtain the intermediate

representation z_t . We then pass z_t through a LSTM module with two LSTM layers. The output of the LSTM module is then passed through a fully connected layer with linear activation to obtain the output logits, $o_t \in \mathbb{R}^{100}$. To obtain the one-hot approximation of generated MIDI number for the t^{th} timestep, \hat{y}_t^{MIDI} we apply the Gumbel-Softmax relaxation on o_t .

We repeat the entire procedure for T timesteps to generate a sequence of MIDI numbers, $\hat{Y}^{MIDI} = [\hat{y}_1^{MIDI}, \dots, \hat{y}_T^{MIDI}]$ for the given lyrics, $X = [x_1, \dots, x_T]$. Similarly, we generate a sequence of duration values $\hat{Y}^{duration} = [\hat{y}_1^{duration}, \dots, \hat{y}_T^{duration}]$ and a sequence of rest values, $\hat{Y}^{rest} = [\hat{y}_1^{rest}, \dots, \hat{y}_T^{rest}]$ using the other two sub-networks respectively. By aligning the three generated sequences we can assemble the melody for the given lyrics.

Discriminator

The role of discriminator is to distinguish between a real melody and a generated melody. The discriminator is a LSTM-based network to model long term dependencies and provide informative signals for generator updates.

Formally, we are given a lyrics i.e. a sequence of syllables $X = [x_1, \dots, x_T]$ such that $x_i \in \mathbb{R}^{20}$ and a melody (real or generated). As mentioned earlier, in this architecture, we represent a melody by three separate sequences: a sequence of MIDI numbers, $Y^{MIDI} = [y_1^{MIDI}, \dots, y_T^{MIDI}]$ where $y_i^{MIDI} \in \mathbb{R}^{100}$, a sequence of duration values, $Y^{duration} = [y_1^{duration}, \dots, y_T^{duration}]$ where $y_i^{duration} \in \mathbb{R}^{12}$, and a sequence of rest values, $Y^{rest} = [y_1^{rest}, \dots, y_T^{rest}]$ where $y_i^{rest} \in \mathbb{R}^7$.

Algorithm 2: TBC-LSTM GAN discriminator forward pass

```

1 for  $t \leftarrow 1, \dots, T$  do
2    $z_t \leftarrow FC_{relu}(FC_{linear}(y_t^{MIDI}) || FC_{linear}(y_t^{duration}) || FC_{linear}(y_t^{rest}) || x_t)$ 
3    $z_t \leftarrow LSTM(z_t)$ 
4    $logits_t \leftarrow FC_{linear}(z_t)$ 
5 end
```

At the t^{th} timestep, the input to the discriminator is the syllable x_t , MIDI number y_t^{MIDI} , duration value $y_t^{duration}$, and rest value y_t^{rest} . We first embed the MIDI number, duration and rest values by passing them through a fully connected layer

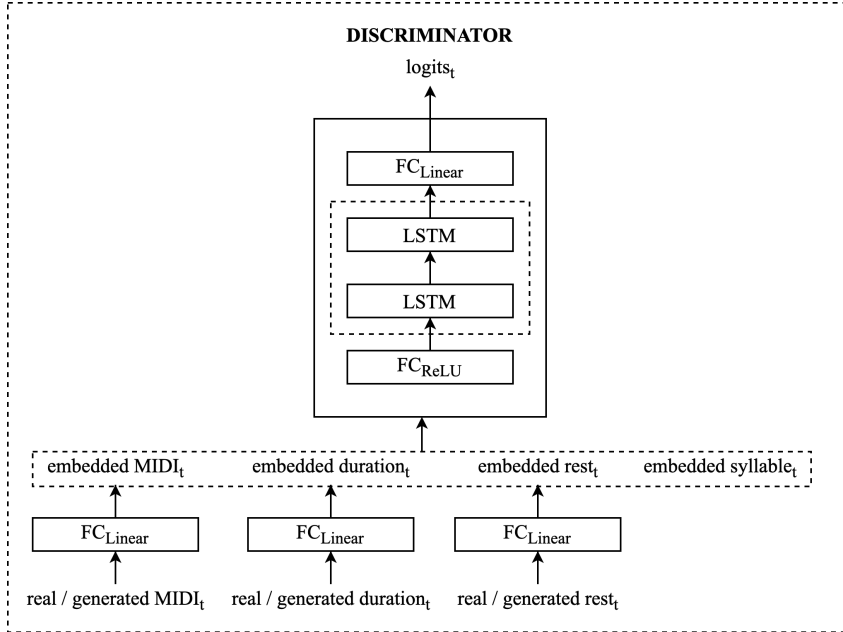


Figure 3.4: Architecture of TBC-LSTM-GAN discriminator

with linear activation. The embedded representation of MIDI number, duration, rest and syllable are then concatenated together to form a joint representation of syllable conditioned music attribute triplet. This joint representation is then passed through a fully connected layer with ReLU activation to obtain an intermediate representation z_t . We then pass z_t through a LSTM module. The output of the LSTM module is then passed through a fully connected layer with linear activation to obtain the output logits, $o_t \in \mathbb{R}$. We repeat the entire procedure for T timesteps to generate a sequence of output logits $[o_1, \dots, o_T]$ such that $o_i \in \mathbb{R}$. We then compute the mean of $[o_1, \dots, o_T]$ and use the average logit for calculating the loss. In Algorithm 2 we provide a succinct description of the forward pass and in figure 3.4, we show the architecture of the discriminator.

3.2.2 Single Branch Conditional (SBC) LSTM-GAN

We know, in our data, a melody is composed of a sequence of triplets of music attributes. To train SBC-LSTM-GAN, we first build a vocabulary of triplets of music attributes. To this end, 3397 unique triplets are extracted from the data. We

can hence denote a melody by $y = [y_1, \dots, y_T]$ such that $y_i \in \mathbb{R}^{3397}$. In figure 3.5 we show the overall architecture of the SBC-LSTM-GAN.

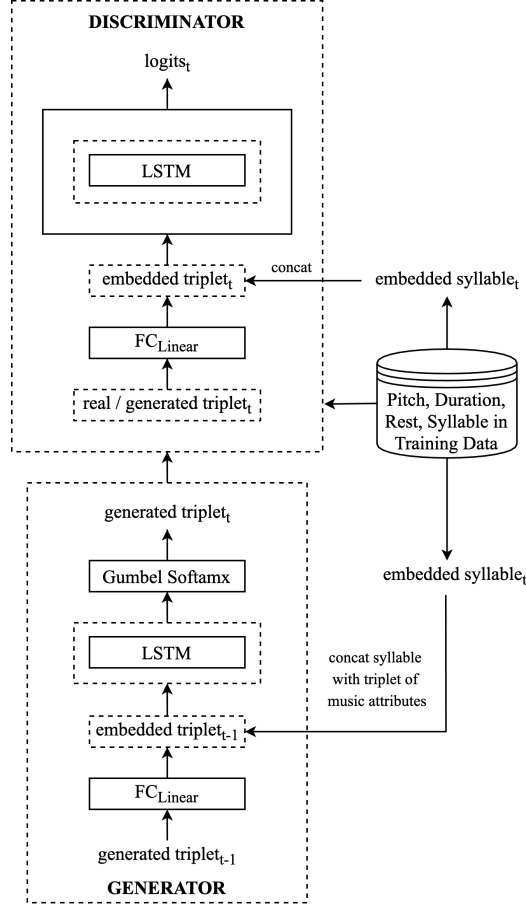


Figure 3.5: Overall architecture of SBC-LSTM-GAN

Generator

Formally, we are given a lyrics i.e. a sequence of syllables $X = [x_1, \dots, x_T]$ and we want to generate a melody i.e. sequence of triplets of music attributes $Y = [y_1, \dots, y_T]$. As mentioned earlier, in this work, we use sequences of length $T = 20$. We know there are 3397 distinct triplets of music attributes present in the data therefore we can represent a triplet of music attributes by a 3397 dimensional one-hot vector. In Algorithm 3 we provide a succinct description of the forward pass and in figure 3.3, we show the architecture of the generator.

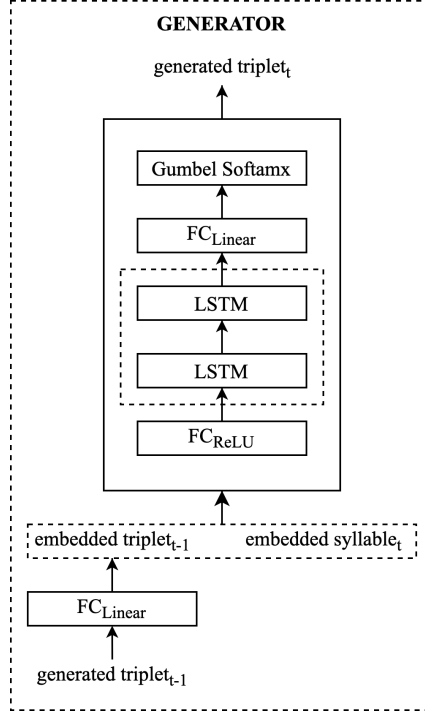


Figure 3.6: Architecture of SBC-LSTM-GAN generator

Algorithm 3: SBC-LSTM GAN generator forward pass

- 1 Sample \hat{y}_0 from $Uniform(0, 1)$
 - 2 **for** $t \leftarrow 1, \dots, T$ **do**
 - 3 $z_t \leftarrow FC_{relu}(FC_{linear}(\hat{y}_{t-1}) || x_t)$
 - 4 $z_t \leftarrow LSTM(z_t)$
 - 5 $\hat{y}_t \leftarrow GumbelSoftmax(FC_{linear}(z_t))$
 - 6 **end**
-

Given a sequence of syllables $X = [x_1, \dots, x_T]$ such that $x_i \in \mathbb{R}^{20}$, we want to generate a melody $\hat{Y} = [\hat{y}_1, \dots, \hat{y}_T]$ such that $\hat{y}_i \in \mathbb{R}^{3397}$. At the t^{th} timestep, the input to the generator is the syllable x_t and the music triplet generated during the previous timestep \hat{y}_{t-1} . To obtain an embedding or dense representation of \hat{y}_{t-1} we pass it through a fully connected layer with a linear activation. We learn the embedding of the triplet of music attributes during the training. We then concatenate the syllable x_t with the embedded triplet representation and pass it through a fully

connected layer with ReLU activation to obtain the intermediate representation z_t . We then pass z_t through a LSTM module with two LSTM layers. The output of the LSTM module is then passed through a fully connected layer with linear activation to get the output logits, $o_t \in \mathbb{R}^{3397}$. To obtain the one-hot approximation of generated music triplet for the t^{th} timestep, \hat{y}_t we apply the Gumbel-Softmax relaxation on o_t . We repeat the entire procedure for T timesteps to generate a sequence of music triplets, $\hat{Y} = [\hat{y}_1, \dots, \hat{y}_T]$ for the given lyrics, $X = [x_1, \dots, x_T]$.

Discriminator

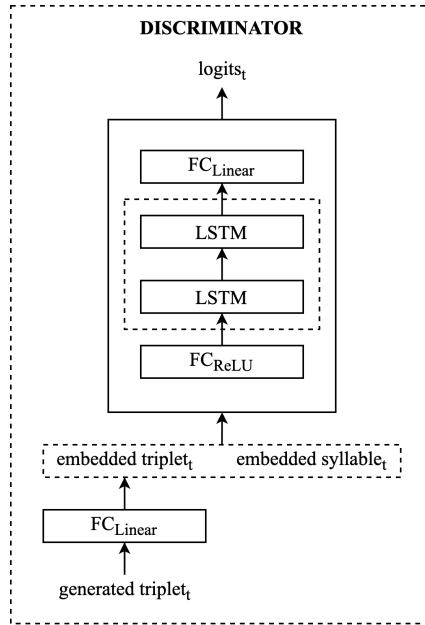


Figure 3.7: Architecture of SBC-LSTM-GAN discriminator

Algorithm 4: SBC-LSTM GAN discriminator forward pass

```

1 for  $t \leftarrow 1, \dots, T$  do
2    $z_t \leftarrow FC_{relu}(FC_{linear}(y_t) || x_t)$ 
3    $z_t \leftarrow LSTM(z_t)$ 
4    $logits_t \leftarrow FC_{linear}(z_t)$ 
5 end

```

Formally, we are given a lyrics i.e. a sequence of syllables $X = [x_1, \dots, x_T]$ such

that $x_i \in \mathbb{R}^{20}$ and a melody (real or generated) i.e. a sequence of triplets of music attributes $Y = [y_1, \dots, y_T]$ such that $y_i \in \mathbb{R}^{3397}$. At the t^{th} timestep, the input to the discriminator is the syllable x_t , and triplet of music attribute, y_t . We first embed the music triplet by passing it through a fully connected layer with linear activation. The embedded representation of the music triplet and syllable are then concatenated together to form a joint representation of syllable conditioned music attribute triplet. This joint representation is then passed through a fully connected layer with ReLU activation to obtain an intermediate representation z_t . We then pass z_t through a LSTM module with two LSTM layers. The output of the LSTM module is then passed through a fully connected layer with linear activation to obtain the output logits, $o_t \in \mathbb{R}$. We repeat the entire procedure for T timesteps to generate a sequence of output logits $[o_1, \dots, o_T]$ such that $o_i \in \mathbb{R}$. We then compute the mean of $[o_1, \dots, o_T]$ and use the average logit for calculating the loss. In Algorithm 4 we provide a succinct description of the forward pass and in figure 3.7, we show the architecture of the discriminator.

Chapter 4

Experiments and discussion

In this chapter, we discuss the experiments performed to show the effectiveness of the proposed model, TBC-LSTM-GAN for lyrics-conditioned melody generation. The data [Yu and Canales, 2019] for our task is composed of a collection of 13,251 melody-lyrics pairs aligned at the note-syllable level. We randomly split the data in the ratio of 8:1:1 to create the train, validation and test sets.

As a baseline for the proposed model, TBC-LSTM-GAN, we use TBC-LSTM-MLE, SBC-LSTM-GAN, SBC-LSTM-MLE, and Conditional-LSTM-GAN [Yu and Canales, 2019]. The TBC-LSTM-MLE and SBC-LSTM-MLE models are the generators of TBC-LSTM-GAN and SBC-LSTM-GAN which are trained with the standard MLE objective.

In this work, we have experimented with two different loss functions during the adversarial training. These include the non-saturating standard GAN [Goodfellow et al., 2014], and the Relativistic standard GAN (RSGAN) [Jolicoeur-Martineau, 2019]. Empirically, we have found the RSGAN function to be the most suitable loss function for our task.

Before we commence the adversarial training, we pretrain the generator for several epochs using the standard MLE objective. During the pre-training stage, we have used the categorical cross-entropy loss function.

Table 4.1: Configuration of the generator and discriminator in TBC-LSTM-GAN

	Branch	Layer 1 : FC(Linear) # units	Layer 2 : FC (ReLU) # units	Layer 3 : LSTM depth	Layer 4 : FC (Linear) # units
Generator	MIDI	128	32	2	64
	Duration	64	16	2	32
	Rest	32	8	2	16
Discriminator	MIDI	128			
	Duration	64	32	2	64
	Rest	32			1

Table 4.2: Configuration of the generator and discriminator in SBC-LSTM-GAN

	Layer 1 : FC(Linear) # units	Layer 2 : FC (ReLU) # units	Layer 3 : LSTM depth	Layer 4 : FC (Linear) # units
Generator	128	64	2	32
Discriminator	128	64	2	32

4.1 Experimental Setup

We use the Adam optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.99$. We perform gradient clipping if the norm of the gradients exceeds 5. Initially, the generator network is pre-trained with the MLE objective for 40 epochs with a learning rate of 0.01. We then perform adversarial training for 120 epochs with a learning rate of 0.01 for both the generator and discriminator. Each step of adversarial training is composed of a single discriminator step and a single generator step. The batch size is set to 512 and a maximum temperature $\beta_{max} = 1000$ is used during the adversarial training. The configuration of generator and discriminator for TBC-LSTM-GAN and SBC-LSTM-GAN is summarized in table 4.1 and 4.2 respectively.

4.2 Music Quantitative Evaluation

To quantitatively measure the melody generated by the proposed model against the different baselines, we compute different music properties [Yu and Canales, 2019]. The music properties are based on MIDI number and temporal music attributes. A melody is a sequence of triplet of music attributes namely MIDI number, duration and rest. Given a melody, we can extract a sequence of individual music attributes and compute the following properties.

1. **MIDI Number Span**: It corresponds to the difference between the largest and smallest MIDI number found in a sequence of MIDI numbers.
2. **3-MIDI Number repetitions** : It corresponds to the number of 3-gram repetitions found in a sequence of MIDI numbers.
3. **2-MIDI Number repetitions** : It corresponds to the number of 2-gram repetitions found in a sequence of MIDI numbers.
4. **Number of unique MIDI numbers** : It corresponds to the number of unique MIDI numbers found in a sequence of MIDI numbers.
5. **Number of notes without rest** : It corresponds to the number of null / zero rests found in a sequence of rest values or the number of notes present in the melody with zero rest.
6. **Average rest value within a song** : It corresponds to the average value of rest given a sequence of rest values i.e. the average period of silence in a given melody.
7. **Song length** : It corresponds to the sum of note duration and rest duration values for a given melody.

Table 4.3: Metrics evaluation of in-songs attributes

	Ground Truth	C-LSTM-GAN	TBC-LSTM-GAN	SBC-LSTM-GAN	TBC-LSTM-MLE	SBC-LSTM-MLE
2-MIDI numbers repetitions	13.54	9.58	11.72	6.52	11.09	6.33
3-MIDI numbers repetitions	5.81	2.28	3.76	1.33	3.17	1.25
MIDI numbers span	10.77	7.65	11.59	13.82	13.86	14.40
Number of unique MIDI numbers	5.88	5.08	5.89	6.86	06.00	06.87
Average rest value within song	0.77	0.63	0.75	0.76	01.17	0.79
Number of notes without rest	15.58	16.70	15.95	15.68	12.72	15.49
Song length	43.25	39.16	42.93	39.62	54.81	43.10

We summarize the results in table 4.3. From the table, it is evident that the proposed model TBC-LSTM-GAN is the closest to the ground-truth for MIDI related song attributes and outperforms other baseline models. For temporal attributes, the proposed model is very close to the ground truth. The proposed model outperforms C-LSTM-GAN [Yu and Canales, 2019] in all properties.

We also visualize the transition between MIDI numbers which is an important music attribute. Figure 4.1 shows the MIDI number transition in the melodies generated by the proposed model, TBC-LSTM-GAN, the ground truth, and baseline models including Conditional LSTM-GAN, TBC-LSTM-MLE, SBC-LSTM-GAN and SBC-LSTM-MLE. As we can observe from the figure and the value of KL divergence (2.16), it is evident that the proposed model best approximates the MIDI number transition in the ground truth.

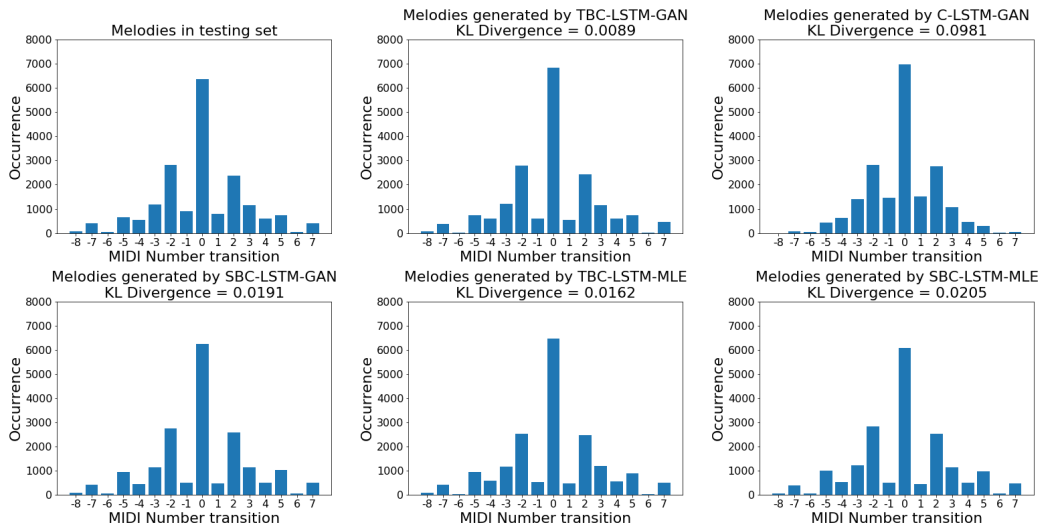


Figure 4.1: Distribution of transitions

4.3 Quality and Diversity Evaluation

In this work, we use the Self-BLEU score to measure the diversity and the MMD score to measure the quality of the generated melodies. Additionally, we perform subjective evaluation to measure the sample quality.

4.3.1 Quality Evaluation

In the figure 4.2, we show the trend of MMD on the testing data during the training. The pre-training and adversarial training stages are separated by the dotted line as shown in the figure. At the start of pre-training, the MMD is high indicating poor

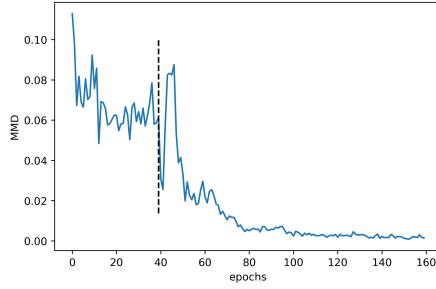


Figure 4.2: Training curves of of MMD scores on testing dataset

quality samples being generated. As we advance into pre-training, the MMD value decreases initially but then stagnates. At the start of an adversarial train, the MMD value initially (upto epoch 50) increases indicating deteriorating sample quality but then continues to decrease (upto epoch 100) indicating improving sample quality. The MMD value then stagnates indicating no change in sample quality.

4.3.2 Diversity Evaluation

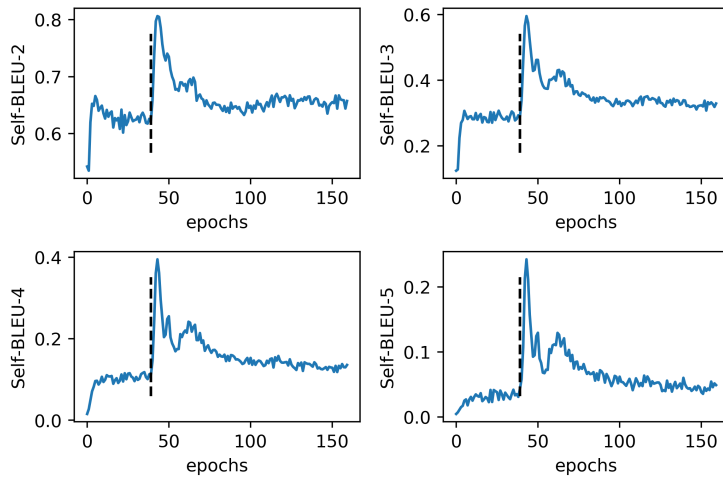


Figure 4.3: Training curves of self-BLEU scores on testing dataset

In the figure 4.3, we show the trend of Self-BLEU score on the testing data during the training. The pre-training and adversarial training stages are separated by the dotted line as shown in the figure. During the pre-training stage, the value of

Self-BLEU does not change much and there is no observable trend. At the start of an adversarial train, the value for Self-BLEU score increases (upto epoch 50) indicates deteriorating sample diversity. However, as the adversarial training progresses, we observe a decreasing trend of Self-BLEU score indicating improved diversity of generated samples.

4.3.3 Subjective Evaluation

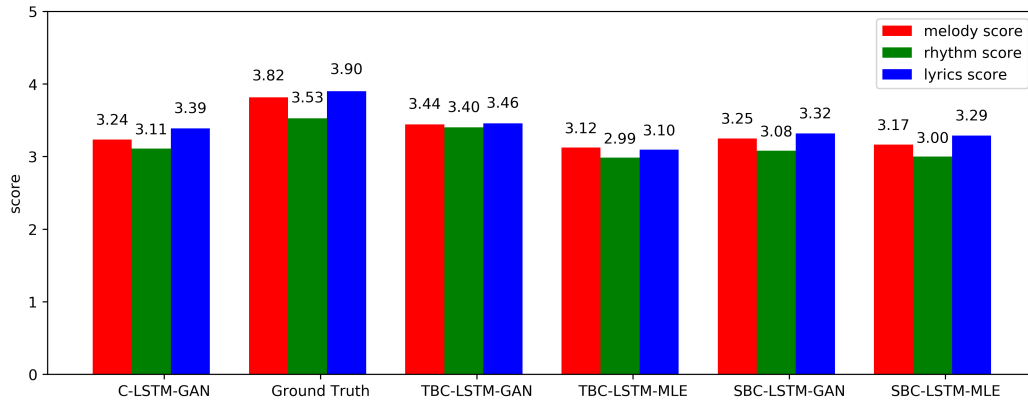


Figure 4.4: Subjective Evaluation Results

We perform subjective evaluation based on the procedure described in subsection 2.8.3 as a means to compare the performance of the proposed model with the baseline models. Results of subjective evaluation are shown in figure 4.4. It is clear from the results that the proposed model, TBC-LSTM-GAN outperforms other baseline models and generates melodies closest to human compositions. Further, we can observe that there exists a gap between the quality of melodies generated by the proposed model and the ground-truth. This indicates that there is room for improvement.

4.4 Generated music

In figure 4.5 we show three examples of generated melody by the trained TBC-LSTM-GAN model using random lyrics from testing set. The lyrics are taken from

songs Run To You (written by Whitney Houston), Catch My Fall (written by Billy Idol), I Love My Friend (written by Andy Williams).

The image displays five staves of sheet music, each with lyrics underneath. The first staff is in bass clef, 4/4 time, and contains the lyrics: "I have the time so I will sing yeah I'm just a boy". The second staff is in treble clef, 4/4 time, and contains the lyrics: "but I will win yeah". The third staff is in treble clef, 4/4 time, and contains the lyrics: "I know that when you look at me there's so much that you just". The fourth staff is in treble clef, 4/4 time, and contains the lyrics: "don't see". The fifth staff is in treble clef, 4/4 time, and contains the lyrics: "She was cry in'and I was lone ly and the band was play in' some old love song".

Figure 4.5: Sheet music for generated melodies

Chapter 5

Conclusion and future work

In this work, we proposed a model, TBC-LSTM-GAN and a baseline, SBC-LSTM-GAN for lyrics-conditioned melody generation. We used a LSTM-based generator to learn the correlation between segments of lyrics and patterns in the melody. We used a LSTM-based discriminator to model long term dependency and provide informative updates to the generator. We exploited the Gumbel Softmax relaxation technique to train a GAN for discrete sequence generation.

Through various experiments we have shown the effectiveness of the proposed model, TBC-LSTM-GAN in generating plausible and tuneful melodies from lyrics. In particular, through quantitative measurements including MIDI related music attributes and MIDI number transition, we have shown that the proposed model best approximates the ground-truth. Also, through subjective evaluation we have shown that the proposed model is the closest to the ground-truth with respect to the quality of generated melodies. Further, we showed how the quality and diversity of generated samples improves as the adversarial training progresses.

We can therefore conclude that the proposed model, TBC-LSTM-GAN outperforms the baseline methods for lyrics assisted melody generation. Also, empirically, we can say that a GAN architecture designed with three branches where each branch is dedicated to a specific music attribute is better than a GAN architecture with a single branch.

Lyrics-conditioned melody generation is a challenging task in terms of design,

implementation and evaluation. As evident from the results of subjective evaluation, our proposed model fails to outperform the ground-truth. Therefore, there is still room for improvement. In this work we work with sequences of length 20 and use Skip-gram models for lyrics encoding. In the future work, we can work with longer sequences and try transformer-based lyrics encoding schemes. Furthermore, we collect and label more data from a variety of music genres to evaluate the robustness of our model. As part of future work, we can explore other interesting related tasks such as melody conditioned lyrics generation.

Bibliography

- [Ackerman and Loker, 2016] Ackerman, M. and Loker, D. (2016). Algorithmic song-writing with ALYSIA. *CoRR*, abs/1612.01058.
- [Bao et al., 2018] Bao, H., Huang, S., Wei, F., Cui, L., Wu, Y., Tan, C., Piao, S., and Zhou, M. (2018). Neural melody composition from lyrics. *CoRR*, abs/1809.04318.
- [Che et al., 2017] Che, T., Li, Y., Zhang, R., Hjelm, R., Li, W., Song, Y., and Bengio, Y. (2017). Maximum-likelihood augmented discrete generative adversarial networks.
- [Chen et al., 2018] Chen, L., Dai, S., Tao, C., Shen, D., Gan, Z., chao Zhang, H., Zhang, Y., and Carin, L. (2018). Adversarial text generation via feature-mover’s distance. In *NeurIPS*.
- [Fedus et al., 2018] Fedus, W., Goodfellow, I. J., and Dai, A. M. (2018). Maskgan: Better text generation via filling in the. *ArXiv*, abs/1801.07736.
- [Goodfellow et al., 2014] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative Adversarial Networks. *arXiv e-prints*.
- [Gretton et al., 2012] Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., and Smola, A. (2012). A kernel two-sample test. *J. Mach. Learn. Res.*, 13:723–773.
- [Guo et al., 2018] Guo, J., Lu, S., Cai, H., Zhang, W., Yu, Y., and Wang, J. (2018). Long text generation via adversarial training with leaked information. *ArXiv*, abs/1709.08624.

- [Hiller and Isaacson, 1958] Hiller, Jr., L. A. and Isaacson, L. M. (1958). Musical composition with a high-speed digital computer. *J. Audio Eng. Soc*, 6(3):154–160.
- [Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.
- [Jang et al., 2016] Jang, E., Gu, S., and Poole, B. (2016). Categorical reparameterization with gumbel-softmax.
- [Jolicoeur-Martineau, 2019] Jolicoeur-Martineau, A. (2019). The relativistic discriminator: a key element missing from standard gan. *ArXiv*, abs/1807.00734.
- [Lin et al., 2017] Lin, K., Li, D., He, X., Zhang, Z., and Sun, M.-T. (2017). Adversarial ranking for language generation. In *Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS’17*, page 3158–3168, Red Hook, NY, USA. Curran Associates Inc.
- [Maddison et al., 2016] Maddison, C. J., Mnih, A., and Teh, Y. W. (2016). The concrete distribution: A continuous relaxation of discrete random variables.
- [Mikolov et al., 2013] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. *arXiv e-prints*, page arXiv:1301.3781.
- [Mirza and Osindero, 2014] Mirza, M. and Osindero, S. (2014). Conditional generative adversarial nets. *CoRR*, abs/1411.1784.
- [Nie et al., 2019] Nie, W., Narodytska, N., and Patel, A. B. (2019). Relgan: Relational generative adversarial networks for text generation. In *ICLR*.
- [Papineni et al., 2002] Papineni, K., Roukos, S., Ward, T., and Zhu, W. J. (2002). Bleu: a method for automatic evaluation of machine translation.
- [Pascanu et al., 2012] Pascanu, R., Mikolov, T., and Bengio, Y. (2012). On the difficulty of training recurrent neural networks. *30th International Conference on Machine Learning, ICML 2013*.

- [Rodriguez and Vico, 2014] Rodriguez, J. D. F. and Vico, F. J. (2014). AI methods in algorithmic composition: A comprehensive survey. *CoRR*, abs/1402.0585.
- [Semeniuta et al., 2018] Semeniuta, S., Severyn, A., and Gelly, S. (2018). On accurate evaluation of gans for language generation.
- [Sutton et al., 2000] Sutton, R., Mcallester, D., Singh, S., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. *Adv. Neural Inf. Process. Syst*, 12.
- [Theis et al., 2015] Theis, L., Oord, A., and Bethge, M. (2015). A note on the evaluation of generative models.
- [Williams, 1992] Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3–4):229–256.
- [Yu et al., 2017] Yu, L., Zhang, W., Wang, J., and Yu, Y. (2017). Seqgan: Sequence generative adversarial nets with policy gradient. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, AAAI’17*, page 2852–2858. AAAI Press.
- [Yu and Canales, 2019] Yu, Y. and Canales, S. (2019). Conditional LSTM-GAN for Melody Generation from Lyrics. *arXiv e-prints, under submission in TNNLS*, page arXiv:1908.05551.
- [Zhang et al., 2017] Zhang, Y., Gan, Z., Fan, K., Chen, Z., Hénao, R., Shen, D., and Carin, L. (2017). Adversarial feature matching for text generation. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70, ICML’17*, page 4006–4015. JMLR.org.
- [Zhao et al., 2018] Zhao, J. J., Kim, Y., Zhang, K., Rush, A. M., and LeCun, Y. (2018). Adversarially regularized autoencoders. In *ICML*.
- [Zhu et al., 2018] Zhu, Y., Lu, S., Zheng, L., Guo, J., Zhang, W., Wang, J., and Yu, Y. (2018). Texygen: A benchmarking platform for text generation models. In *The 41st International ACM SIGIR Conference on Research and Development*

in Information Retrieval, SIGIR '18, page 1097–1100, New York, NY, USA. Association for Computing Machinery.