MANCHESTER
1824
The University of Manchester

Neural Music Generation

author: Weilue Luo
supervised by: Dr. TingTing Mu

April 30, 2021

Abstract

This report discuss implementations and evaluations of various proposed techniques used in deep learning for neural music generation. It includes RNN-based and Attention-based models. It investigated the possibilities of combinations of the above techniques and corresponding performance analysis were conducted through visual and comparative means. In addition, properties and effects of the input data were also explored for both monotonic and polyphonic representation. There are also attempts on CNN-based model. Variants of the implementing techniques were also proposed in attempt to search for potential improvements.

Acknowledgement

I would like to express my gratitude towards Dr. TingTing Mu for guiding me into fascinating world of machine learning in my year 2 course as well as useful feedback on the direction of this project and report. I would also like to thank Yian Deng for giving me helpful pointers. I could have not completing this project without their support.

Covid-19 Impact

The quality of the work is slightly impacted as I could not collect enough feedback from the public to assess performance of the models.

Preface

Music is is one of the universal culture of the human society. With advancement of technologies in recent decades, music is becoming more and more accessible, it brings much entertainment to our everyday life. Today, we can enjoy different kinds of music through the internet and many are inspired to become music composer. In my childhood when listening to different music, I often wonder if I could create all these wonderful pieces one day. Of-course this quickly become a disillusionment due to my tone deafness, but I hope to full-fill my childhood dream in a different way through this project.

Contents

Chapter 1

Introduction

1.1 Aims & Objectives

Neural music generation is a diverse field, there are many novel techniques being proposed every now and then, my direction on this report would be to try out various methods and have a general understanding of the current state of neural music generation. This report will roughly follows the following structure:

1. Computational creativity and the brief history.
2. Overall organization and common techniques used.
3. Overview of each sub-field under the organization and detailed analysis of a few notable work in each sub-field.
4. The approach of my work and the experiment I have done.
5. Evaluation & Conclusion.

Neural music generation refers to generate musical intermediate representation, for example score or the binary file format *midi* [1]. Today neural music generation has been an active research area and has a number of sub-fields, e.g. style transfer, music in-painting, audio synthesis and score-to-audio. (Ji et al., 2020) provide a good review on the overall progress of music generation in different areas. They divide music generation into three different levels according to different intermediate representation, namely score generation, performance generation and audio generation , Figure 1.1 shows the overall structure.

Throughout this report, the focus will be on monophonic and polyphonic generation, which fall between score and performance generation under this organization. But techniques from audio generation is also attempted in search for improvement. I will investigate and compare the performance of the embedding, residual connection and attention mechanism have on the traditional RNN network. Various architecture based on the attention mechanism and transformer are also compared and attempts on the convolution-based architecture is also described.

1.2 Terminologies

In term of formula, **x** denotes input to some function, **w** denotes learnable weight matrix, **y** denotes output of some function, n denotes the size of the input, $i \in n$ denotes index of one of the input, x^T denotes the

[1] https://www.midi.org/specifications/file-format-specifications/standard-midi-files

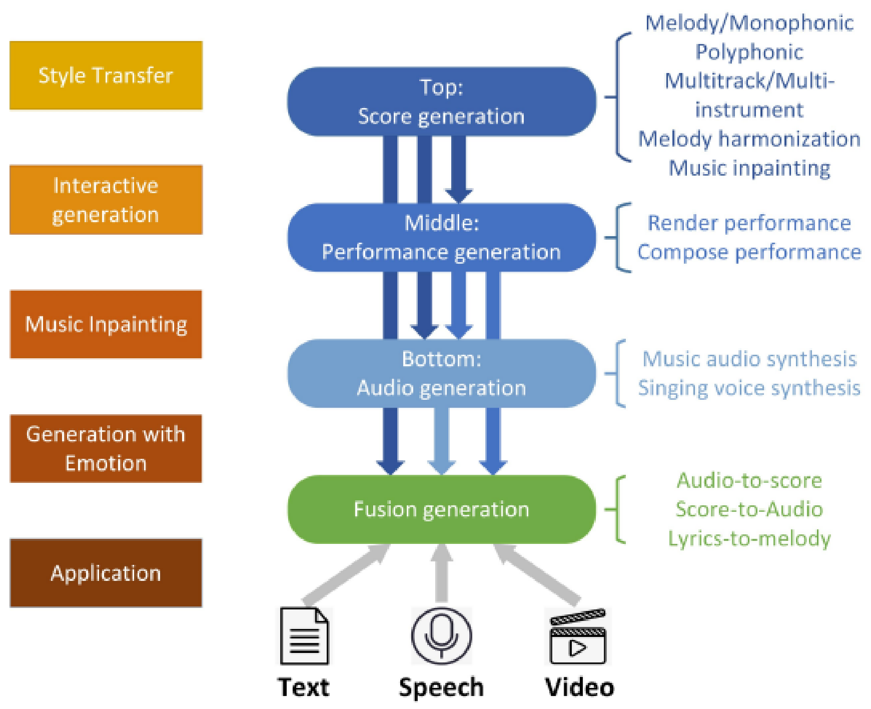


Figure 1.1: left side represents music generation tasks that are not limited to one level; middle represents the three levels of music generation; arrow denotes lower level result can be conditioned on the higher level; right side denotes specific generation tasks in each level; green level indicate that the three levels can be fused together and combine with other content.
source: `https://arxiv.org/pdf/2011.06801.pdf`

transposition of x, the bold $\mathbf{h}$ denotes some form of hidden information, p denotes probability, $\mathbb{E}$ denotes expectation, and $||x||_2^2$ denote the squared l_2 norm of x.

In term of ML terms, *dense* layer refers to fully connected or linear layer, *RNN* refers to recurrent neural network, h denote hidden state of LSTM cell, c denote the cell state of LSTM cell, λ denote hyper-parameters, b denotes the bias, and $\mathbf{z}$ denote latent variable (usually in some form of encoder output).

Chapter 2

Background

While highly skilled music composers are able to produce plausible compositions, the question of whether machine can behold creativity had much debate, the evolution of neural music generation has been quite interesting. This chapter will cover the background of music generation, including a brief overview from computational creativity to the history of neural music generation, as well as technical background and terminologies used in the report.

2.1 Computational Creativity

Music generation is a multidisciplinary field that tries to achieve computational creativity. But our usual experience with computer is that machine behaves deterministic-ally, same output when given for the same input, this seems to oppose the idea of creativity, in fact, Ada Lovelace[1], one of the inventor of the earliest computer (analytical engine), once made a remark on Babbage's[2] Analytical Engine[3]:

> The Analytical Engine has no pretensions whatever to originate anything. It can do whatever we know how to order it to perform. It can follow analysis; but it has no power of anticipating any analytical relations or truths. (Lady Lovelace)

Many years later Alan Turing [4] reduce her opinion to assertion of "computer cannot surprise human" and argue that computer can still surprise human, especially when the consequence is not immediately recognizable (Turing, 1950); later some also stated that although the programmer needs to follow computer grammar, he is not restricted to express content, he can have a set of acceptable answers, write down the rules of selections or give computer advice about making a decision (Minsky, 1968). But the answer to whether computer can demonstrate human level creativity is still an interesting area for us to explore (Carnovalini and Rodà, 2020).

2.2 Brief History

Music generation is one of the prolific task of computational creativity, and algorithmic music generation has more than 60 years of history. There were a number of attempts to generate music algorithmic-ally

[1]https://en.wikipedia.org/wiki/Ada_Lovelace
[2]https://en.wikipedia.org/wiki/Charles_Babbage
[3]https://en.wikipedia.org/wiki/Analytical_Engine
[4]https://en.wikipedia.org/wiki/Alan_Turing

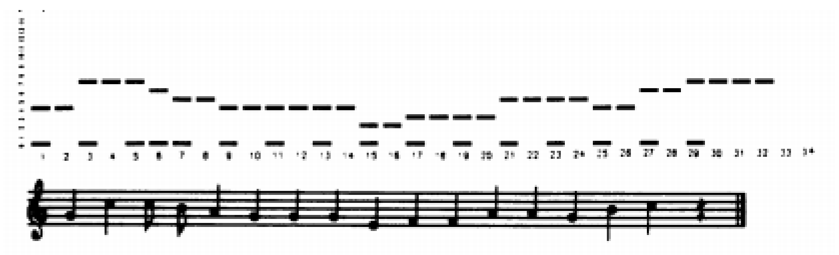


Figure 2.1: Two linear layers model output, interpolate between two melodies
source: `https://www.jstor.org/stable/pdf/3679551.pdf`

(Ji et al., 2020). During late 1960s, a French music composer named Lannis Xenakis[5] made use of probability distributions to generate music through manipulate digital samples (Luque, 2009). Later in 1989, some researchers begin to reproduce melodies from musical pieces using neural nets (Bharucha and Todd, 1989), he showed that a model consists of two linear layers was able to model and capture the *schematic expectation* [6] of monophonic sequence, Figure 2.1 shows an sample output.

But it was only up till recent years, the ease of access to more computational power give raise to machine learning, which showed promising signs in various specialised domains (Sengupta et al., 2019), it was hoped that it can demonstrate creativity to a certain extent due to its black box nature. Nowadays, review on creativity mechanism of deep neural network suggest that neural networks can perform a number of tasks that we human consider creative (Wyse, n.d.), for example, neural style transfer on image (Gatys et al., 2015). But there are also limitations, for example, the most effective AI algorithms still rely on supervised learning, whereas true creativity processes do not have pre-defined outcome (Anantrasirichai and Bull, 2021).

[5]`https://en.wikipedia.org/wiki/Iannis_Xenakis`

[6]Human brain's predictive mechanism to music, e.g. when you hear "do-re-mi-fa", you will have expectation of hearing "sol" as the next octave. https://en.wikipedia.org/wiki/Melodic_expectation

Chapter 3

Prior Knowledge

In this chapter, I will be describing some prior knowledge that are necessary for understanding the recent works on and my experiments on music generation. Note that not all techniques are used in my experiment, namely vector quantization, VQVAE, relative attention and short-time fourier transform. I will try to explain each of the concept in a concise way but some basic concept of machine learning is required (which a typical computer science third year student would have).

3.1 Representation

3.1.1 Score

Score is a guide to perform a piece of music, It shows which and when to play each note, relative to each other; timing information is provided through the implicit relative metrical grid and in some cases, time signature [1]. Figure 3.1 shows an example of the musical score.

Note that the score does not tell you how to play the music exactly. The time signature only tells you the relative speed between quarter notes; and dynamics [2] is often quite arbitrary. For example, ***P*** and ***PP*** in figure 3.1 means play quietly and play very quietly respectively, but it neither tell us how quiet nor if all notes are equally quiet. Therefore in music generation it is more commonly stored as some score-like representation that allow for more precise information, like midi.

[1]A notation used in music theory to specify how many beats (notes) are contained in each measure (bar), `https://en.wikipedia.org/wiki/Time_signature`

[2]How music get louder or quieter. `https://en.wikipedia.org/wiki/Dynamics_(music)`

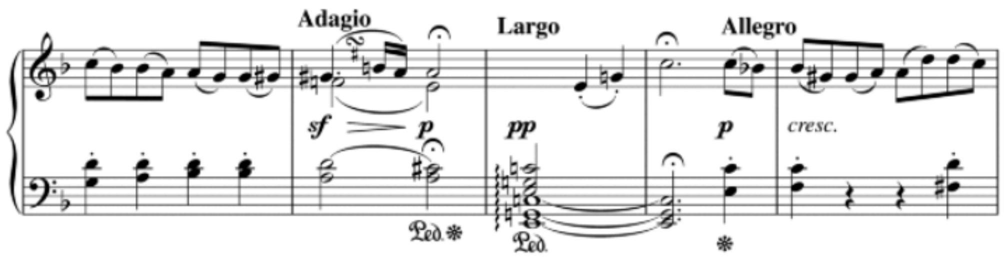


Figure 3.1: Excerpt from the “Tempest”, Piano Sonata Op.31 No.2, L.van Beethoven
source: `https://www.virtualsheetmusic.com/score/Tempest.html`

3.1.2 Midi

In everyday life, music we encounter usually stored in raw audio formats which are convenient for end-user (e.g. wav, mp3, flac, etc...), but it is not an ideally compact format to store musical information. In the field of professional musical composer and neural music generation, you often see another format, named *midi* [3] (*The MIDI Association - Standard MIDI Files*, n.d.). Formally, midi is a technical standard that used as a communication protocol, it is designed to facilitate communications between musical instruments. In simple term, it stores when and which instrument is being play at what note, time and momentum, in an event-like representation, called midi messages. The ease of manipulations and modification allows researchers to conduct studies at higher level of abstraction. In this report, unless specifically stated, the format for any dataset is midi, as it is commonly used in score and performance generation task. Using the midi file format has a number of benefits, for example it can effectively reduce the size of the input and allow the model to interpret on the higher level of abstraction; however, it also limits the model to only a number of pitches with some fixed durations of timestep and pitches which may hurt creativity.

There are many other researches that utilities different other formats, such as audio generation which generate the raw waveform, which I will also introduce in later section of the report.

3.1.3 Dynamics

In the context of piano performance, the term dynamic refers to the speed of the performer hitting a particular key, which simply translate to amplitude of audio. It maybe a surprise but it was only until recent years that researchers started looking into generating music with various dynamic (Oore et al., 2018). Before that, researchers often preprocess the training data without including the amplitude information and post-process the generated music with a fixed, pre-defined, sense-able amplitude.

3.2 Techniques

3.2.1 Long Short-term Memory

Traditional Recurrent Neural Network (RNN)[4] often suffer from the gradient vanishing problem. This is because the unrolled RNN is often a very deep network and deep network is intrinsically unstable (Nielsen, 2015). Another reason is that in the unrolled RNN the same weight in the RNN cell is used for all time-steps, this means that there is a long chains of multiplication which can easily leads to gradient vanishing/exploding problem.

LSTM (Hochreiter and Schmidhuber, 1997) is a type of RNN cell that tries to solve the above problems. It consists of three gates that serves different functionalities, they utilize the hidden state that keep track of information to modify the cell state. The forget gate (denotes by the first multiplication of the input cell state) is one of the gate in LSTM, it allows the cell to decide what is important and should not be forgotten, this controls the gradient value at each time-step and update the parameters accordingly (The other gates also played an important role but this is the main idea, (*Understanding LSTM Networks – Colah's Blog*, n.d.) provides an more in-depth explanation on how it works.). An overview of LSTM cell is shown in figure 3.2.

[3] `https://www.midi.org/specifications/file-format-specifications/standard-midi-files`

[4] `https://en.wikipedia.org/wiki/Recurrent_neural_network`

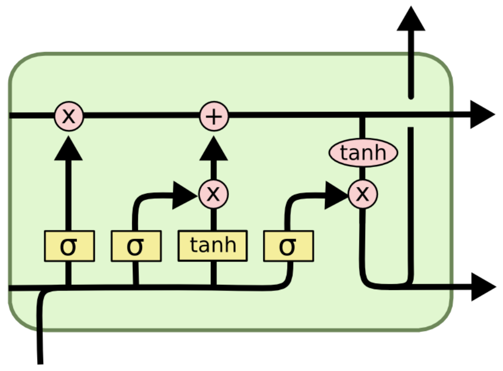


Figure 3.2: Overview of LSTM. σ denotes the sigmoid function , tanh denotes the tanh function, joining line denotes concatenation and spliting line denotes replication.

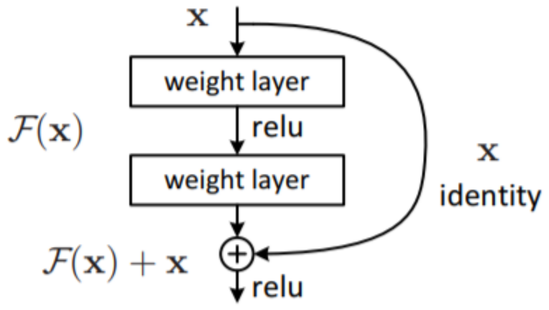


Figure 3.3: Overview of the residual connection mechanism.
source: `https://openaccess.thecvf.com/content_cvpr_2016/papers/He_Deep_Residual_Learning_CVPR_2016_paper.pdf`

There are also other variants that tried to solve the vanishing gradient problem of LSTM, for example, the gated recurrent unit (Cho et al., 2014).

3.2.2 Residual Connections

The residual connections (or skip connections) (He et al., 2015) is a mechanism that first applied in image recognition task, which won the ImageNet 2015 (Deng et al., 2009) challenge. Residual connection allows gradient to flow through the network directly without passing through the activation, it has the benefit of reducing the chance of gradient explode/vanish due to nature of non-linearity of the activation. Some people argue that skip-layer connections should not improve on the model performance, because it is just a linear term which the model could model it easily. But the real benefit of residual connection is it provides an alternating path for gradient, since the long chain of multiplications is the root cause of gradient explode/vanish. Figure 3.3 shows an overview of the residual connection mechanism.

As pointed out by (He et al., 2015), it is important to keep the identity of the residual network, which means the residual should not be applied to any operation, they also shown that non-identity connections can result in gradient vanishing problem.

3.2.3 Attention & Multihead Attention

Attention

Attention (Bahdanau et al., 2016) is a term that first used in machine translation that extends the existing encoder-decoder RNN-based architecture by allowing the model to search for parts in the source sentence that are relevant to the translation to the target sentence. Surprisingly, this attention mechanism was shown to be far more useful than just machine translation, in particular, *self-attention* (or intra-attention), a type of attention that relate different part of a sequence and compute a representation, has been found to be successfully applied into many tasks (Vaswani et al., 2017).

Multihead Attention

Multi-head attention simply means to perform attention multiple times with individual projection [5] of Q K, and V, then concatenate and project again to form a output (Vaswani et al., 2017). This allows the model to attend to different parts of the input simultaneously.

$$\text{multi-head attention}(Q,K,V) = \text{dense}_1(\text{concat}([\text{head}_1, \text{head}_2, \dots, \text{head}_n])) \tag{3.1}$$

$$\text{head}_i = \text{attention}(\text{dense}_2(Q), \text{dense}_3(K), \text{dense}_4(V)) \tag{3.2}$$

The generality of multihead attention mechanism allowed it to be widely applicable to many tasks, including music generation.

3.2.4 Scaled dot-product attention & Relative Attention

Scaled Dot-product Attention

Scaled dot-product attention can be mathematically described as follows:

$$\text{scaled dot-product attention} = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.3}$$

where Q, K, and V are query, key and value respectively, d_k is the dimension of key and query. It is *scaled* because it is divided by $\sqrt{d_k}$, this is a factor that scale down the result of matrix multiplication, this works better for large dimension because as the dot product increase in magnitude, the differences between elements in the resulting matrix gets larger, therefore pushing the result of softmax function to have extremely small gradient for some elements.

Note that what query, key, and value represents depends on the context, in self-attention, all three are referring the same input vector; in the case of multi-head attention used in the decoder stage in transformer, key and value are the latent vector **z** (commonly referred as memory) and query refers to the output of the previous masked multi-head attention (i.e. learning which part of the latent vector **z** should the model attend to).

[5] A forward pass of the dense layer

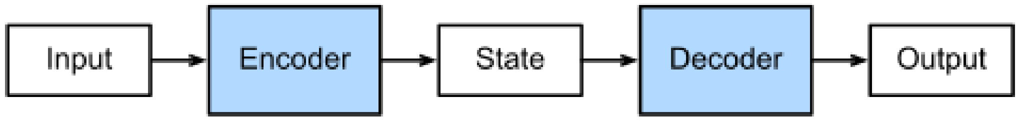


Figure 3.4: The encoder-decoder architecture.
source: `https://d2l.ai/chapter_recurrent-modern/encoder-decoder.html`

Relative Attention

Relative attention (Shaw et al., 2018) is a type of attention that based on scaled dot product self attention, it introduce embedding to the attention mechanism, which is added to the projected K. The embedding represents the distance between each pair of word, and it was shown to improve the BLEU[6] (a evaluation metric for machine translation task) score compared with the vanilla transformer. Equation below shows the implementation of relative dot-product attention:

$$\text{Relative Scaled dot-product attention} = \text{softmax}\left(\frac{Q(K + \text{embedding})^T}{\sqrt{d_k}}\right)V \tag{3.4}$$

In the usual transformer architecture (which will be described later), the positional information is absolute and hard-coded, which maybe suitable in some cases, but for music generation, relative positioning makes more sense in term of rhythm and beats.

3.2.5 Encoder & Decoder

The encoder-decoder architecture is first introduced in (Sutskever et al., 2014) for sequence-to-sequence machine translation task. It showed that deep LSTM model with encoder-decoder architecture can outperform the traditional statistical machine translation (SMT) based system. Figure 3.4 shows a general diagram for the encoder-decoder architecture.

3.2.6 Transformer

The transformer model follows the encoder-decoder architecture and make use of the attention mechanism, where the encoder maps the input $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ to a continuous latent variable $\mathbf{z} = [z_1, z_2, \ldots, z_n]$ and the decoder map $\mathbf{z}$ to an output sequence $\mathbf{y} = [y_1, y_2, \ldots, y_3]$ one at a time. At each step the model is *auto-regressive*, which simply means the model uses the previous generated symbol as the current input when generating the current symbol. The model mainly made use of a combination of two types of attention: *scaled dot-product attention* (see sec. 3.2.4) and *multi-head attention* (see sec. 5.11).

The transformer consists of two parts, encoder and decoder. Figure 3.5 shows a detailed organization of the transformer architecture.

3.2.7 Sparse Transformer

Sparse transformer (Child et al., 2019) is a variant of the transformer architecture. It introduced several sparse factorisation of the attention matrix (a.k.a. factorized attention) which scale down the computation from $O(n^2)$ to $O(n\sqrt{n})$. Authors of the paper found that the learned attention patterns are common in some ways as shown in figure 3.6.

[6] `https://en.wikipedia.org/wiki/BLEU`

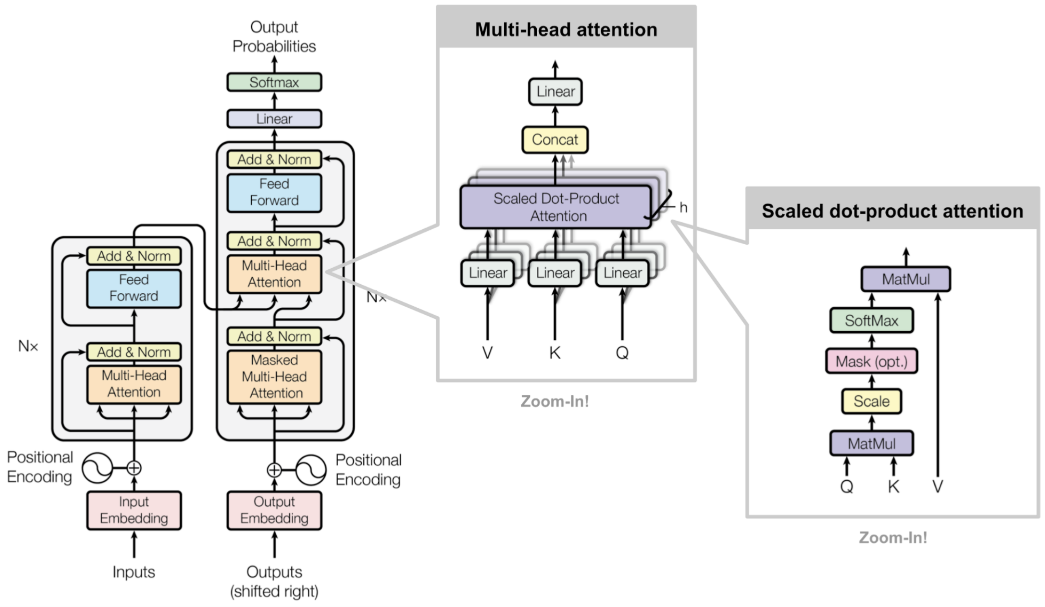


Figure 3.5: The overview of the transformer architecture.
N denotes the number of stacks.
Norm denotes the layer normalization.
Positional Encoding denotes the absolute positioning of the token in the input sequence (details in the source).
Feed forward denotes two linear layers with relu activation in between.
The term masked is used to denote hiding of the future token so that the model cannot predict the future token itself when predicting the future token.
source: `https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html`

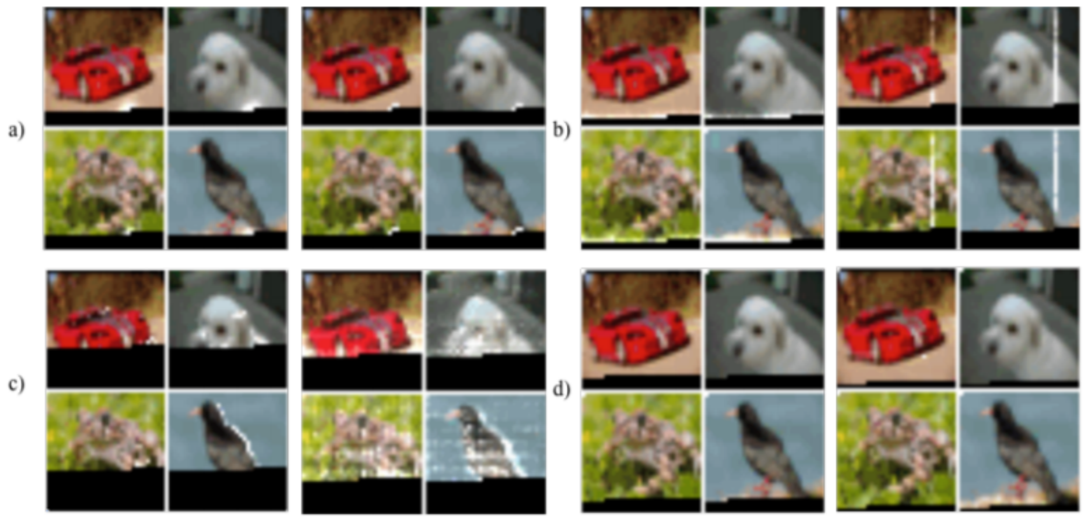


Figure 3.6: Learned attention patterns. white highlight denotes attention weight while generating a pixel; black denotes autoregressive mask.
a) attenion on nearby pixels.
b) attention on vertical and horizontal pixels.
c) global data-dependent attention.
d) highly sparsed attention.
source: `https://arxiv.org/pdf/1904.10509.pdf`

Their experiment showed that the model are able to learn specialised sparse structures, which may be the reason why they are applicable to different domains, but most of the pattern only make use of a small amount of location, which means this may not be the best architecture. Therefore they introduced different form of attention as shown in figure 3.7.

This architecture only attends to some of the previous data, the gap between each attended position are carefully chosen so that it is close to $\sqrt{n}$, which significantly reduce the computational cost. Note that by stacking attention layers, it also allow information from other positions to be summarised, although the model may not be able to access it directly. This sparse attention mechanism is useful for data that has a periodic structure, and therefore yielded SOTA result on many tasks related to images and music.

3.2.8 Auto-encoder & VAE

Auto-encoder

Auto-encoder (Hinton, 2006) is a type of unsupervised learning architecture that attempts to learn a efficient representation of the given data, typically used for dimensionality reduction (See section 3.2.12) and feature learning[8]. It is often used for eliminating redundant information or noise in the raw data. A typical architecture includes an encoder which compress the given data into a compact representation; on the other side, a decoder is also learned to reconstruct the original data using the compressed representation. Note that to perform the task perfectly would mean that to copy the input to the output, typically when the decoder is overpowered, leading to a phenomena called posterior collapse (Lucas et al., 2019) where the learned representation is served as a dictionary key instead of useful information, therefore it

[7]`https://en.wikipedia.org/wiki/Adjacency_matrix`
[8]`https://en.wikipedia.org/wiki/Feature_extraction`

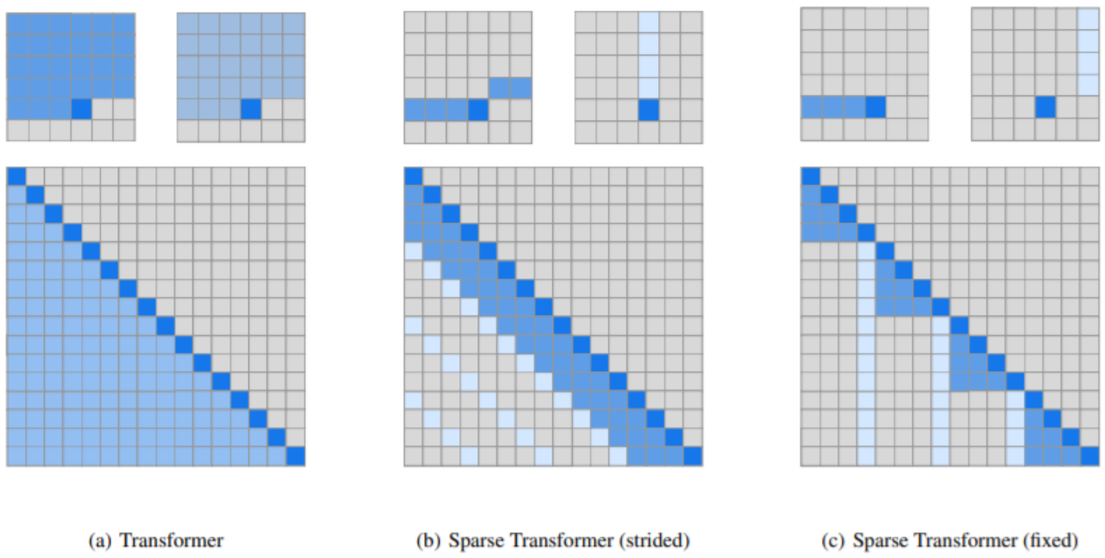

(a) Transformer
(b) Sparse Transformer (strided)
(c) Sparse Transformer (fixed)

Figure 3.7: The top row shows which position of the attention head are received for the input. The bottom row shows connectivity matrix[7](not to scale).
source: `https://arxiv.org/pdf/1904.10509.pdf`

is often restricted to force to reconstruct original data approximately.

VAE

VAE (Kingma and Welling, 2014) refers to variational auto-encoder. *Variational* refers to a type of auto-encoder that learns the representation through a continuous distribution space rather than a discrete space, allow for a more natural representation. Although VAE has proven to be an effective model for producing semantically meaningful latent representation, it was found to be limited to sequential data and models at that time and have difficulty modelling sequences with long-term structure, therefore often not directly used in music generation.

3.2.9 Quantization

Numerical data can be quantized into a number of predefined bins to reduce the size of representation, in other words, it maps values from a large set (often continuous) to a smaller set (often countable and finite). Common subject of quantization in neural music generation includes time-step and velocity.

Quantization is common in many works in music generation and my experiments. A typical 3 minutes midi audio file can contain around $1,000$ notes, each note has varying duration of infinite precision. If we restrict the minimum unit of the duration to 0.1 second, then it will produce around $2,000$ encoded data, which is only a double of the original notes, however, this often distort the original audio by a large amount, later in my experiment, I decided to choose 0.02 as the minimum unit, this will produce around $10,000$ samples. To further demonstrate the differences, below are links to two audio tracks, quantize and no quantize.

- Monophonic un-quantized sample. `https://soundcloud.com/kallzvx/no-quantize-sample/s-MZ1OB7xhAwg`

- Monophonic quantized sample (minimum time-step is 0.02 second). `https://soundcloud.com/kallzvx/quantized-sample/s-lWHT6RZL2ZN`

3.2.10 Vector Quantization & VQVAE

Vector Quantization

Vector Quantization (VQ)[9] is a quantization technique that originally used for data compression, it divides points in the input vectors to a number of groups and iteratively update each point by moving them closer to the closest centroid point, in the simplest form, it picks the next updating point at random.

VQVAE

VQVAE (van den Oord et al., 2018) stands for vector quantized variational auto-encoder. It learns a discrete representation by mapping the output of the encoder to the nearest element in an embedding space, which circumvent the posterior collapse issue and yields similar result compared to a typical VAE.

Note that the gradient cannot pass through the operation of finding the closest point. In the original paper, the gradient from the decoder is skipped through the operation, and passed directly to the encoder output. Then they update the embedding space by using the l_2 loss to move the embedding vectors towards the encoder output as shown below:

$$\lambda||z_e(\mathbf{x}) - \text{sg}[e]||_2^2 \tag{3.5}$$

where λ is a hyperparameter that set to 0.25[10]; z_e denote the latent output from the encoder; e denote the embedding vector and sg denote the stop gradient operator which defined as identity during forward and has zero partial derivative.

3.2.11 Batch & Layer Normalization

Batch Normalization

Batch normalization (Ioffe and Szegedy, 2015) is a commonly used techniques to avoid internal co-variance shift of the input by regulating the input mean and variance of the layer input. Not only it helps to reduce the need for dropout, but also allow higher learning rate without worry about divergence. It update the input as follows:

$$\text{BatchNorm}(\mathbf{x}) = \lambda_1\hat{\mathbf{x}} + \lambda_2 \tag{3.6}$$

$$\hat{\mathbf{x}} = \frac{\mathbf{x} - \mu_B}{\sqrt{\sigma_B^2 + \varepsilon}} \tag{3.7}$$

$$\simeq \frac{\mathbf{x} - \text{mean}(\bar{\mathbf{x}})}{\text{std}(\bar{\mathbf{x}})} \tag{3.8}$$

where λ_1 and λ_2 denote the trainable matrix and bias respectively, μ_B denotes the mean of the input batches , σ_B^2 denotes the variance of the input batches, ε is a very small constant added for numerical stability and $\hat{\mathbf{x}}$ is the input summed over different batches.

[9] `https://en.wikipedia.org/wiki/Vector_quantization`

[10] The author of the original paper did not notice a difference by varying it between 0.1 and 2.0.

Layer Normalization

Layer normalization (Ba et al., 2016) is a technique similar to batch normalization. But $\hat{\mathbf{x}}$ is the sum of all inputs over single batch instead of sum of one input over different batches. This improves the training time and generalization of existing RNN and transformer models, which is commonly used in music generation.

3.2.12 Dimensionality Reduction

The large amount of data used to represent music in different audio format often trouble researchers in the field of neural music generation, this is because computer does not have enough computational power to process [11] so much data. For example, a typical raw audio WAV [12] file stores audio sampled at $44,100$ Hz with 16 bits per sample, this means that for one second of an audio, $705,600$ bits are required to encode its information, it would be difficult for model to learn from such large input, therefore dimensionality reduction is commonly used in music generation to decrease the size of the input, it can also be applied to encode information in a more discrete representation. Dimensionality reduction can come in various form:

- μ-law encoding (Brokish and Lewis, n.d.). This is originally used in digital communication, reducing the dynamic range of the audio signal and allow audio to be transmitted with increased signal-to-noise ratio (SNR) [13].
- Autoencoder & VAE. (see sec. 3.2.8)
- Quantization. (see sec. 3.2.9)

In this project I also used the midi file format (See section 3.1.2) instead of the raw audio approach, to further reduce the size of the input.

3.2.13 Gradient Clipping

Gradient clipping is a common technique to clip the size of the gradient to prevent the model to aggressively updating the parameter near sharp area on the loss surface, in this report's experiment, it is simply done by clipping the parameter gradient in an element-wise manner before the parameter gets a update.

3.2.14 Short-time Fourier Transform

Short-time Fourier Transform (STFT)[14] is a fourier-related[15] transform used to determine the frequency and phase of a local segment of a long signal. The procedure is to divide the long audio segment into equal length and apply Fourier Transform separately; and fourier transform[16] is a function that decompose space/time-depending functions to spatial/temporal frequency, for example, express musical chord as volume and frequencies or its constituent notes.

[11] Here process refers to training of the neural network, we do have the computational power to store, manipulate and play the file.

[12] `https://en.wikipedia.org/wiki/WAV`

[13] `https://en.wikipedia.org/wiki/Signal-to-noise_ratio`

[14] `https://en.wikipedia.org/wiki/Short-time_Fourier_transform`

[15] `https://en.wikipedia.org/wiki/Fourier_analysis`

[16] `https://en.wikipedia.org/wiki/Fourier_transform`

3.3 Miscellaneous

3.3.1 Sound Font

Sound font[17] is commonly refers to as a file format and its associate technology which uses synthesis to convert midi files to audio. Different sound font can result in different audio to be generated from the same midi file. In this report, the most common sound font I used is the Essential Keys-sforzando sound font[18], which is a typical soundfont for piano.

[17] `https://en.wikipedia.org/wiki/SoundFont`
[18] `https://sites.google.com/site/soundfonts4u/`

Chapter 4

Related Works

In this chapter, I will be presenting some of the previous generation methods using neural network, including both generating from scratch and generating with conditions/constraints such as chord and lyrics. Much technical details will be discussed inline with the work to aid understanding. For each music generation category introduced by (Ji et al., 2020), I will also discuss in depth for two works which I considered most representative, and some of their introduced techniques are experimented.

4.1 Score Generation

There are numerous work on score generation. In 2016, (Bretan et al., 2016) shows that combination of autoencoder and deep structured semantic model with LSTM can outperform stacked LSTM models. A more famous example would be (*Generating Long-Term Structure in Songs and Stories*, n.d.) from Google Magenta, it includes a basic RNN model and two RNN model variants that designed to generate long term structure. There are also works that attempts to use GAN for generation, vanilla RNN-based GAN is not suitable for discrete token generation due to two problems: (1) it is hard for the gradient to flow back to the generator; and (2) the discriminator can only assess on the complete generated sequence. inspired by reinforcement learning, (Yu et al., n.d.) introduced a new way to update the generator: they use the discriminator as the reward function, and Monte Carlo Tree search to approximate the state-action reward for each generated token in the complete sequence for back-propagation. In 2017, (Yang et al., 2017) also explored the probability of using CNN-based GAN to generate melody in the symbolic domain. In this section, I will be focusing on introducing Melody RNN (Bretan et al., 2016) and MidiNet (Yang et al., 2017).

4.1.1 Melody RNN

Melody RNN (*Generating Long-Term Structure in Songs and Stories*, n.d.) is a project in Google back in 2016 that attempts to explore the possibility of generating sequences such as melodies with long-term structure in midi events. There are two types of models available: **Lookback RNN** [1] and **Attention RNN** [2], both of these models are built upon the Basic Mono RNN model. Basic Mono RNN is simply a one-hot vector of the previous event and predict the following event using LSTM cells.

[1] `https://github.com/magenta/magenta/blob/master/magenta/models/melody_rnn/melody_rnn_model.py#L165`

[2] `https://github.com/magenta/magenta/blob/master/magenta/models/melody_rnn/melody_rnn_model.py#L179`

Lookback RNN is similar to Basic RNN, except additional information and label are added to the input vector.

- Events from 1 to 2 bars before the previous bar are added, allows the model to recognize pattern more easily.
- Information of whether the previous event is repeating is added, this signal allows the model to recognise repetitive state more easily.
- Fixed time signature of 4/4 are also added as binary step clock:
  - step 1: `[-1,-1,-1,-1, 1]`.
  - step 2: `[-1,-1,-1, 1,-1]`.
  - step 3: `[-1,-1,-1, 1, 1]`.
  - step 4: `[-1,-1, 1,-1,-1]`.

Attention RNN uses the attention mechanism to learn even longer term structure, it works by allowing the model to learn the way to condition on previous steps. Note that the attention mechanism used in this model is different from what we known today, specifically:

$$u_i^t = v^T \tanh(w_1 h_i + w_2 c_t) \tag{4.1}$$

$$a_i^t = \text{softmax}(u_i^t) \tag{4.2}$$

$$h_t = \sum_{i=t-n}^{t-1} a_i^t h_i \tag{4.3}$$

where the vector v and matrices w_1 and w_2 are learn-able matrix parameters, and h_i and c_i are the output of the current RNN hidden states and cell states respectively. The u_i^t represents how much attention each previous step t should receive and softmax is used to normalise the values (for all possible i). An attention mask is also created to weight each of the previous step. The h_t is then concatenated with the RNN cell output follow by a linear layer and create the final output of the RNN cell, feeding to the next RNN cell.

Even though both models are limited to monophonic piano music, and often fall into a repeating hole and not very rhythmic, Melody RNN is able to generate music more consistent throughout a long generated sequences than most previous attempts. With some editing, some simple tracks can be made. You can find generated samples and more information in their blog post [3].

4.1.2 Midi Net

Many models for music generation use RNN, but since the success of Wave Net, there were also attempts to explore the possibility of convolutional network. One of the model that produced notable result is Midi Net, it attempts to generate melody using GAN architecture, where both generator and discriminator are convolutional network.

[3] `https://magenta.tensorflow.org/2016/07/15/lookback-rnn-attention-rnn`

Architecture

The representation of the music is represented by a h-by-w matrix, where h denotes the number of notes and w denotes the number of time steps, and each entry $\mathbf{x} \in \{0,1\}^{h \times w}$ denotes whether the note at the particular time step has been turned on (omitting velocity).

The core of Midi Net is deep convolutional generative adversarial network (DCGAN), it trains two networks, the generator G which attempts to generate fake samples with random input (noise) to "fool" the discriminator D, and D will learn to distinguish real and fake samples, by challenging each other, we hope after some training, G can generate samples that are indistinguishable from the real. Mathematically, GANs learn G and D by solving:

$$\min_{G}\max_{D} V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}[\log(D(\mathbf{x}))] + \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}[\log(1-D(G(\mathbf{z})))] \tag{4.4}$$

where $\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}$ denotes expectation of sampling from real data and $\mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z})}$ denotes expectation of sampling from random distribution.

Regularization Techniques

Due to the nature of minimax game, training of GANs are often unstable and subject to many issues such as *mode collapsing*. Mode collapsing refers to a phenomena where generator produce a small set of plausible outputs that discriminator cannot distinguish, and generator may learn to produce only that small set of outputs (local minimum), then the discriminator's best strategy is to memorise that small set of output, now since the discriminator only remember only that small set of outputs, when the next iteration comes, the generator can easily produce plausible results. As a result the generator will only rotate through a set of outputs, and this is called mode collapse. Among various technique, Midi Net applied *feature matching* and *one-sided label smoothing* to the training process to avoid these effects.

Feature matching (Salimans et al., 2016) refers to adding a L2 regularisers as follows:

$$\lambda_1 ||\mathbb{E}\mathbf{x} - \mathbb{E}G(\mathbf{z})||_2^2 + \lambda_2 ||\mathbb{E}\text{conv}_1(\mathbf{x}) - \mathbb{E}\text{conv}_1(G(\mathbf{z})||_2^2 \tag{4.5}$$

where conv_1 denotes the first convolutional layer in the discriminator and λ_1 and λ_2 are hyper parameters that weight the contribution. We can easily deduce that by adding the above loss, the model will try to match the distribution of real and generated data for both the generator output and the first convolutional layer output.

One-sided label smoothing (Salimans et al., 2016) is used to avoid overconfidence of the discriminator, this means that the discriminator only use a subset of the input feature to detect real samples, and the generator is then in turn only generate these features since others are not used by the discriminator, what one-sided label smoothing does is that it penalise the discriminator when the prediction goes over 0.9, this can be done easily by setting the output true label to be 0.9 instead of 1.0.

The generator consists of 4 layers of transposed convolutional layer, each conditioned on the chord to generate corresponding melody, the chord condition is also feed into 4 layers of convolution, each of the output are concatenated to the input before feeding into the transposed convolution. Transpose convolution is originally used in CNN network for up-sampling, you can think of it as reverse operation of the convolution layer. The discriminator is simply two convolutional layers follow by a dense layer, its input also has the chord condition, there is also a 1D condition that encode any prior knowledge such as musical scale and profiles of the melody.

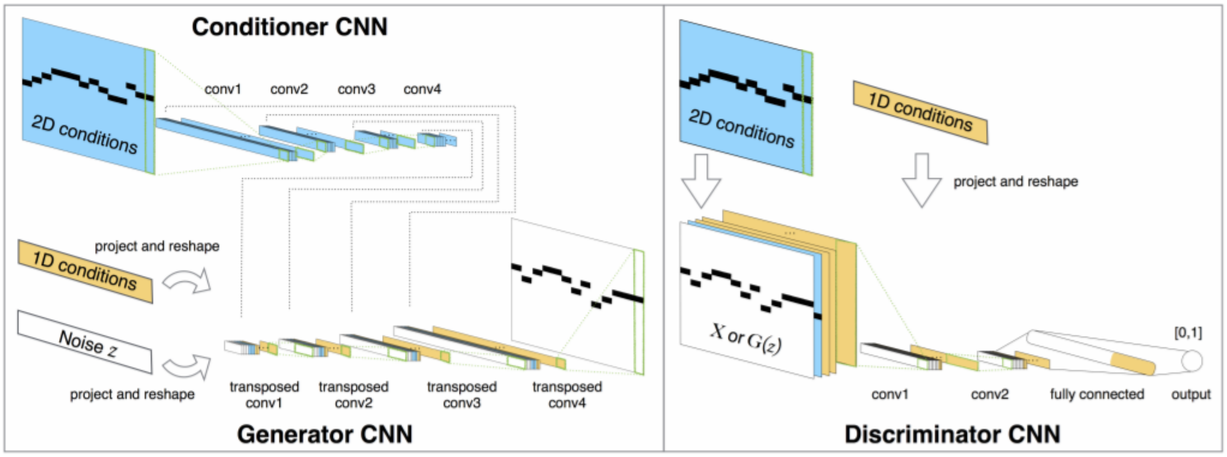


Figure 4.1: System diagram of the proposed MidiNet model for symbolic-domain music generation. source: `https://arxiv.org/pdf/1703.10847.pdf`

To trade-off between creativity and discipline, one can use only part of the intermediate 2D chord condition or decrease the values of λ_1 and λ_2 to control the feature matching restriction to give more freedom to the model.

Result

In the experiment, they compare the performance of the models by finding people to rate their samples, the result shows that MidiNet performs comparably with MelodyRNN models in being pleasant and realistic while being much more interesting. Moreover, their model is only trained on 500+ melody tabs while the MelodyRNN model was trained on thousands of samples, though the exact number were not disclosed.

MidiNet 1 denote the model with only melody conditioning, and *MidiNet 2* denotes the model with additional chord conditioning, and the *Lookback* and *Attention* model are the MelodyRNN models that I introduced before. In summary, MidiNet showed that convolution-based GAN are able generate melody, its flexible architecture also allow easy extension and primer conditions to be introduced. However, its fixed velocity also limiting the model to have more natural output.

4.2 Performance Generation

Score generation is similar to human generation, and there is no exact information on timing and dynamics as we discussed, on the other hand, performance generation refers to adding timing and dynamic information to score generation, giving more flexibility to the model. There are also other previous work that yield notable result, for example, (*CVAE Piano Performance Rendering - Demo*, n.d.) first proposed a LSTM-based VAE and later further extended his work to condition the input on the positional dependent information, and it was shown to be able to model the piano performance; and (Jeong et al., 2019) proposed to use Graph Neural Network (Scarselli et al., 2009) to model music with a specific type of graph then render performance using it. In this section, I will be focus on presenting two most notable work in recent year in the area of performance generation, namely performance rnn and music transformer.

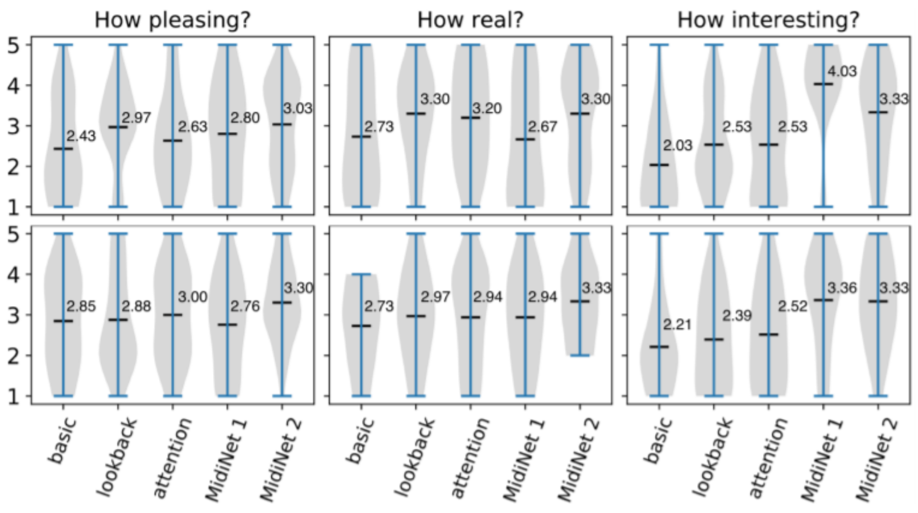


Figure 4.2: Result of a user study comparing MelodyRNN and MidiNet models
source: `https://arxiv.org/pdf/1703.10847.pdf`

4.2.1 Performance RNN

Performance RNN(*Performance RNN: Generating Music with Expressive Timing and Dynamics*, n.d.) is a LSTM-based generative model that designed model polyphonic music with expressive timing and dynamics.

Representation

It uses an event-based representation, which consists of four different types: *note on*, *note off*, *time-shift* and *velocity*. Here, velocity is the event that encodes the dynamics of the music and time-shift encodes the duration until the next event occur. They use 128 note on and note off events, corresponding to the 128 midi pitches; 100 time shift event that encode the duration from 10 ms up to 1 second, moving forward in time to the next event; and 32 velocity events, which quantized the midi velocity into 32 bins, they changes the following velocity of the event. These add up to 388 events and input to the network as one-hot representation.

Implementation

The overall network is comparatively simple, they used three LSTM layer with size 512, dropout applied to the output of each LSTM, finally a linear layer take the resulting sequence. They used the Yamaha e-Piano Competition dataset, which contains midi files from around 1400 professional piano performer. Data augmentation using time stretching is also applied to the input data (making each performance 5% faster or slower). You can find more information and samples in their blog. post[4].

Conclusion

They shown that the model is able to learn some dynamics information by judging on the generated samples beginning with various styles, which is promising, however, although the model is able to learn

[4]`https://magenta.tensorflow.org/performance-rnn`

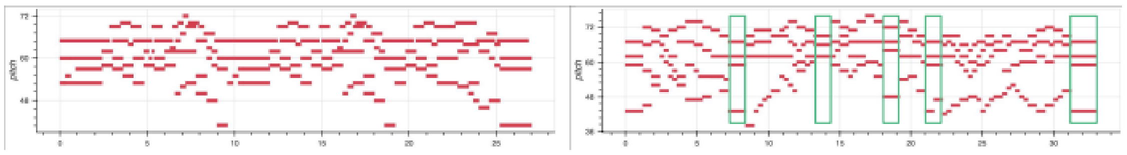


Figure 4.3: Sample of pianoroll from vanilla transformer (left) and transformer with relative self attention (right). Green boxes indicate where the correct timing is held.
source: `https://arxiv.org/pdf/1809.04281.pdf`

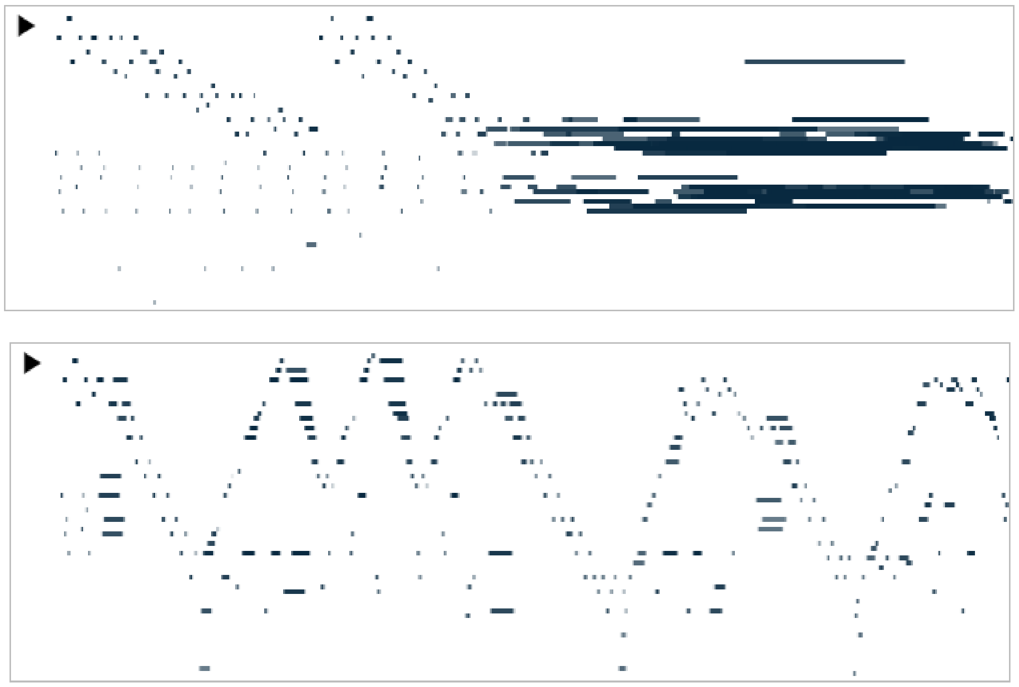


Figure 4.4: Comparison of one of the sample from vanilla transformer and the transformer with relative self attention.
source: `https://arxiv.org/pdf/1809.04281.pdf`

about the structure of the representation, the model itself has much misinterpretation on the long-term structure.

4.2.2 Music Transformer

Music transformer (Huang et al., 2018) is a transformer-based model that utilize the relative self attention mechanism (see sec. 3.2.4) and shown to be able to generate long term coherent music at the consistent style.

I will skip the details of the representation and implementation, because the representation used was same as the performance RNN (see sec. 4.2.1) and the implementation is same as the vanilla transformer (see sec. 3.2.6) except it is using relative attention (see sec. 3.2.4)[5]. Therefore I will just give a short description of their experiment and result here.

[5]There are more fine details, e.g. using a more efficient implementation of the relative attention, but it would be too tedious to introduce it here

They used the J.S. Bach chorales dataset[6] for evaluation, they see that relative attention drastically improve the lossw over the baseline transformer and was reflected on the sample quality. Samples were shown to maintain the necessary timing/instrument grid, that is, it always learn to complete 4 note events before using a time-shift event. Figure 4.3 shows a comparison between the models. Since the local timing is consistent, the model is able to capture timing on a global level giving raise to regular phrasing. In figure 4.4, we can see that the transformer with relative self attention is able to keep a consistent pattern through out long generated music, while the vanilla transformer's quality deteriorate after some time.

In conclusion, music transformer demonstrate that transformer with self attention is able to capture long-term structure and it is well-suited for music generation task, which provides an exciting direction for research. You can find samples and more information from their blog post[7].

4.3 Audio Generation

Audio generation, as its name suggests, it generates the raw audio waveform, which is most intuitive and easily perceived form of representation. The most notable work would be Wave Net, it showed that dilated convolution (discussed later) is able to generate realistic speech and music pieces (though lacking long-term consistency); later (Engel et al., 2017) also created a VAE based on Wave Net and produced expressive sound; and (Donahue et al., 2019) also shown GAN can generate audio that is more preferred by human and much faster in order of magnitudes than the original Wave Net; more recently, JukeBox is also proposed and shown that combining auto-encoders with up-samplers and conditional lyrics information can generate music with singing. In this section, I will focus on the Wave Net and Juke Box, as the former is one of the most representative break-through in recent years and the later represents the SOTA.

4.3.1 Wave Net

WaveNet (van den Oord, Dieleman, Zen, Simonyan, Vinyals, Graves, Kalchbrenner, Senior and Kavukcuoglu, 2016), WaveNet is a paper that brought many attention back in the days, because it is one of the earliest model that operate on raw audio while achieve SOTA in the field of text-to-speech audio generation, before WaveNet, researches are often not done on raw audio, since the number of samples in raw audio is too large for the available computational resource. Although it was not very effective in music generation from today's point of view, its technique has inspired many music generation works even up until today.

Architecture

WaveNet is based on PixelCNN (van den Oord, Kalchbrenner, Vinyals, Espeholt, Graves and Kavukcuoglu, 2016), which is a work explores conditional image generation, WaveNet predict the next sample similar to PixelCNN models the joint probability distribution of each pixel i over an image x:

$$p(\mathbf{x}|\mathbf{h}) = \prod_{i=1}^{n^2} p(x_i|x_1,\ldots,x_{i-1},\mathbf{h}) \tag{4.6}$$

where h represents the prior information, $x_i,\ldots,x_{i-1}$ denotes the pixels before x_i, and x_i represents the predicting pixels, and n denotes the length of the image (assuming the image is square).

[6]https://github.com/czhuang/JSB-Chorales-dataset

[7]https://magenta.tensorflow.org/music-transformer

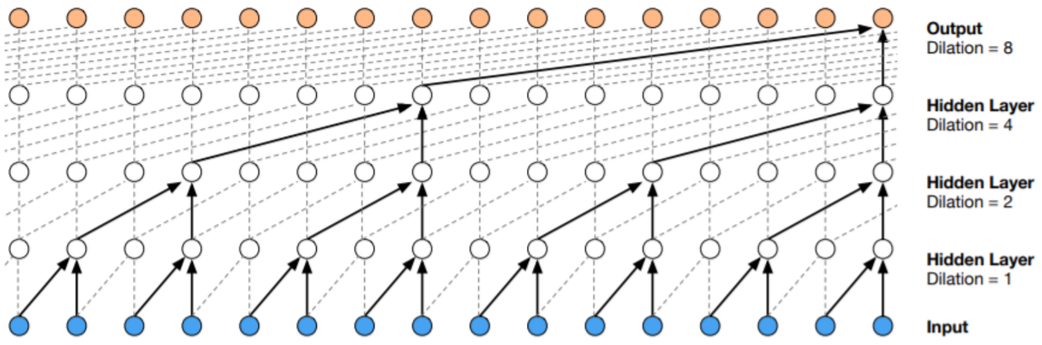


Figure 4.5: Visualization of a stack of dilated causal convolutional layers

WaveNet applied this idea and uses the dilated causal convolution. The term *causal* simply refers to the assumption that all data points before a specified window is not relevant, as convolution layers required a consistent input size and does not use up all historical data. The term *dilated* refers to a mode of operation in the convolution layer where its filter is applied over an area larger than its specified kernel size by skipping on the input values, figure 4.5 shows an overview of the casual dilated convolution.

Since the dilated convolutions doubles for each added layer, this means the receptive field grow exponentially, allowing the network to have larger capability which is crucial for generating musical audio, and stacking these blocks further increases the capacity and receptive field size.

WaveNet also adapted the *gated convolutional layers* used in PixelCNN (van den Oord, Kalchbrenner, Vinyals, Espeholt, Graves and Kavukcuoglu, 2016), that is, replacing rectified linear activation units between masked convolutions with following gated activation unit. This technique is commonly used in RNN model to control what kind of information is allowed to let through, the multiplicative operations allows the model to model more complex interaction and avoid vanishing gradient, and this technique is found to be better than ReLU in this case.

$$\mathbf{y} = \tanh(w_{\text{filter}} * \mathbf{x}) \odot \sigma(w_{\text{gate}} * \mathbf{x}) \tag{4.7}$$

where w_{filter} denote the weight of the filter. w_{gate} denote the weight of the gate, σ denotes the sigmoid non-linearity, $\odot$ denotes the element-wise multiplication, and $*$ denotes convolution operation, this is effectively same as the original gated activation unit used in PixelCNN, except that it does not have the hidden information h.

Lastly, since each raw audio sample are often stored in 16-bit representation, this requires the model to predict one of $65,565$ classes, to make things more tractable, they also applied *μ-law encoding* [8] to the samples and quantise them to 8 bits. μ-law encoding is a technique originally used in digital communication, it reduces the dynamic range of the audio signal and allow audio to be transmitted with increased signal-to-noise ratio (SNR) back in the days where there is not much bandwidth. Figure 4.6 shows an overview of the wavenet block.

Result

Although quantitative evaluation of these models are difficult, but a subjective evaluation was conducted by listening to the samples produced. The researchers found that the large receptive field was crucial

[8] `https://en.wikipedia.org/wiki/%CE%9C-law_algorithm`

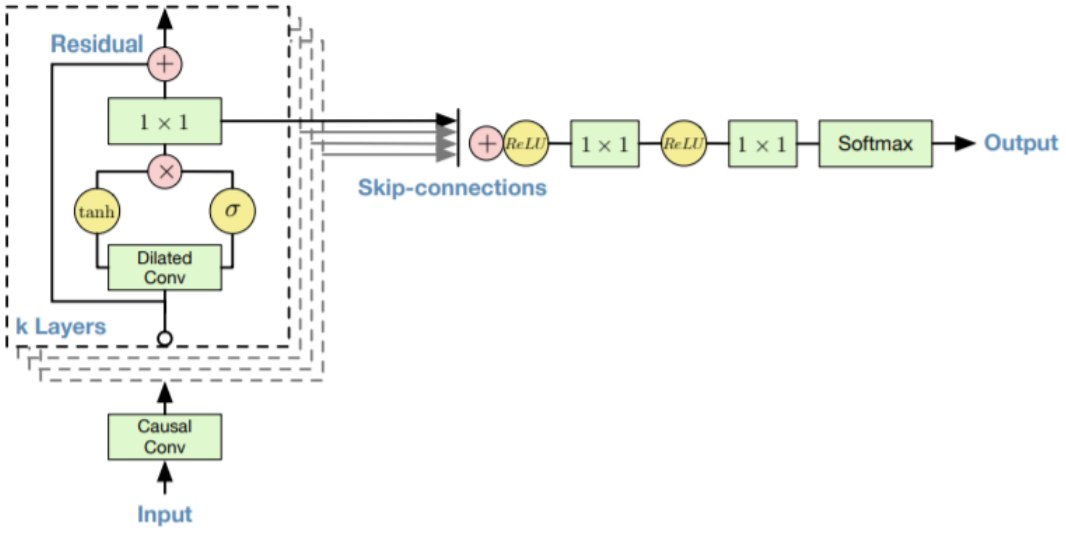


Figure 4.6: Overview of the WaveNet block.
source: `https://arxiv.org/pdf/1609.03499.pdf`

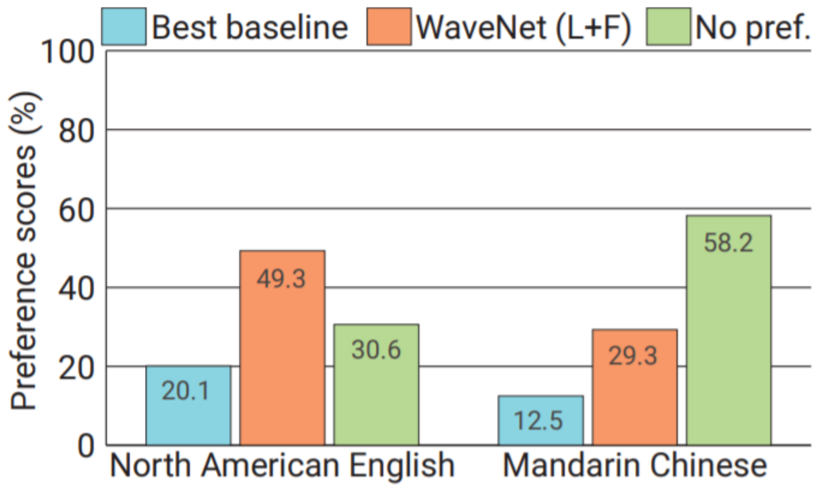


Figure 4.7: Subjective preference score of samples from best baseline of LSTM-RNN-based statistical parametric and HMM-driven unit selection concatenative baseline synthesizers, and WaveNet that conditioned on both linguistic features and logarithmic fundamental frequency.
source: `https://arxiv.org/pdf/1609.03499.pdf`

for it to sound musical, but the samples were not able to enforce long-term consistency and coherence. Nevertheless, the samples produced by the model were often harmonic and aesthetically pleasing. Their experiment result shows that WaveNet is able to produce result that has preference score higher than the best baseline from LSTM-based and HMM-driven synthesizers in both English and Chinese music. Figure 4.7 shows subjective preference comparison of models.

To conclude, WaveNet presents a deep auto-regressive generative model that combine causal and dilated convolutions to model long ranged temporal dependency, and showed promising results when applied to music generation, but the overall architecture is not able to produce a coherent piece of music. More information can be found in their blog [9].

4.3.2 JukeBox

JukeBox (Dhariwal et al., 2020) is a autoregressive generative model released by OpenAI[10] that is able to generate highly diverse and fidelity songs. It is also capable of singing along unseen lyrics and coherence up to multiple minutes.

Approach

For the input audio, they first applied an multi-scale VQVAE (See section 3.2.10), *multi-scale* refers to using three different levels to encode the input, varying the amount of information retained. Next, three prior models are trained to learn the distribution of music codes encoded by the VQVAE, generating and mapping the compressed, high level codes to the lower level codes, finally, the uncompressed code is passed to the lowest level VQVAE decoder to generate music.

Music VQVAE

The VQVAE in Jukebox applied a wavenet-style non-causal convolutions, interleaving with down/up-sampling to match varying input length. They also made a number of modifications shown below to avoid a number of problems:

- **Codebook Collapse** is a known problem of VQVAE(Dieleman et al., 2018), to overcome this, the authors of jukebox also applied *random restarts*, where when the utilization of the latent code fall below a certain threshold, the content of codebook is reset to the current batch of the embedding output.
- **Poor Utilization of Upper Level Encoder**. They trained three auto-encoder separately to maximize the amount of information stored in each level.
- **Attention to Low Frequency Only**. The prior models were found to capture the low frequencies only during reconstruction, so they also introduced a loss that encourages the model to math the spectral components regardless of how hard to learn:

$$\mathcal{L} = |||\text{STFT}(\mathbf{x})| - |\text{STFT}(\hat{\mathbf{x}})|||_2$$

  where STFT denotes the Short-time Fourier Transform (See sec. 3.2.14), $||x||_2$ denotes the l_2 norm of x and $\hat{\mathbf{x}}$ denote the mean frequency.

[9] `https://deepmind.com/blog/article/wavenet-generative-model-raw-audio`
[10] `https://openai.com/`

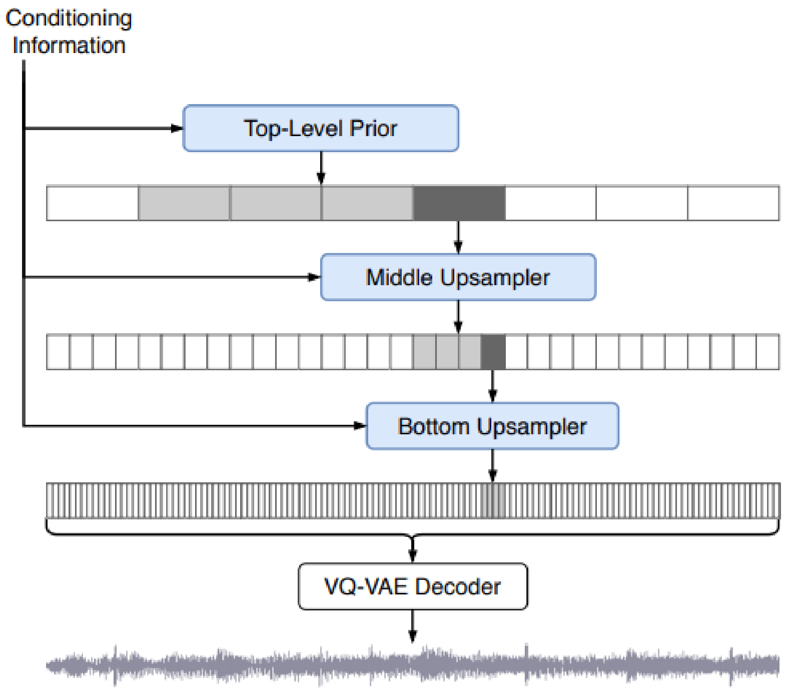


Figure 4.8: Overview of how three upsamplers work together to upsample the discrete codes (generated by VQVAEs) and pass to the bottom level VQVAE decoder.
source: `https://arxiv.org/pdf/2005.00341.pdf`

Upsamplers

After we have the Music VQVAE to generate codes, we need to train the upsamplers to learn the distribution of the prior, which separated into three auto-regressive models:

$$\begin{aligned} p(\mathbf{z}) &= p(\mathbf{z}^{\text{top}}, \mathbf{z}^{\text{middle}}, \mathbf{z}^{\text{bottom}}) && (4.8) \\ &= p(\mathbf{z}^{\text{top}})p(\mathbf{z}^{\text{middle}}|\mathbf{z}^{\text{top}})p(\mathbf{z}^{\text{bottom}}|\mathbf{z}^{\text{middle}}, \mathbf{z}^{\text{top}}) && (4.9) \end{aligned}$$

where $p(\mathbf{z}^{\text{top}})$, $p(\mathbf{z}^{\text{middle}}|\mathbf{z}^{\text{top}})$ and $p(\mathbf{z}^{\text{bottom}}|\mathbf{z}^{\text{middle}}, \mathbf{z}^{\text{top}})$ corresponding to the three different upsamplers. Each upsampler consists of transformer with sparse attention[11]. To condition on external information such as lyrics and genres, they uses a deep residual wavenet(Xie et al., 2017), followed by transpose convolution and layer-norm to encode these information, which is then concatenate with the discrete code from VQVAE and passed to the current level upsampler, figure 4.8 shows an overview of the upsampler architecture.

Result

The dataset used was over 1.2 million songs, the overall model consists of more than 5 billions parameters, and was trained for more than 1 month on over 512 V100 GPUs. The resulting samples are evaluated manually, the researchers found that the model can generate samples stays coherent, frequently imitate musical harmonies able to produce diverse samples. The samples also shown to be able to align lyrics

[11]They also proposed to use an similar implementation called Scalable Transformer which does not required a specialized CUDA-kernel

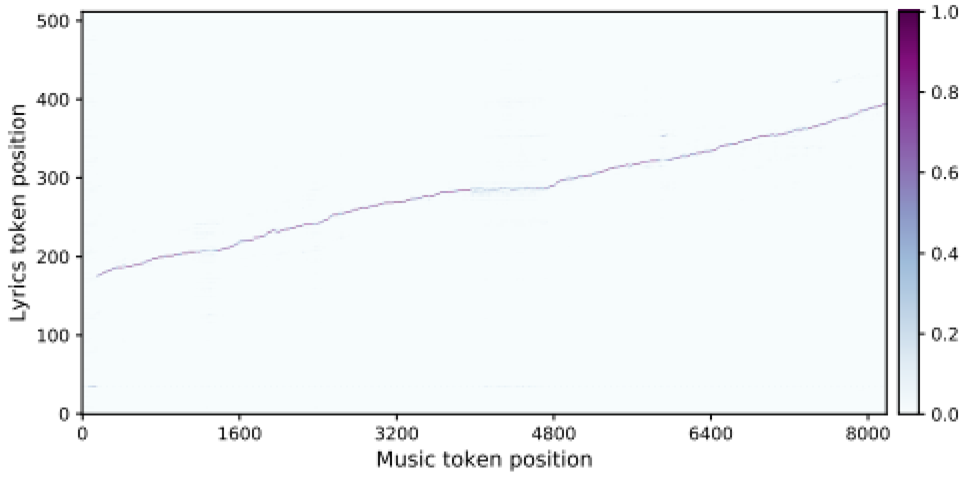


Figure 4.9: Lyrics and singing alignment learned by one of the attention layer in the network, x-axis shows the position of the query and y-axis shows the position of key.
source: `https://arxiv.org/pdf/2005.00341.pdf`

and singing using the attention mechanism, figure 4.9 shows the learned alignment by one of the attention layer.

While the work shows a step forward to generate coherent long raw audio music samples, but it has a number of problems: firstly, it takes very long to generate samples, 1 minute of raw audio requires 1 hour to generate, as the upsampler process the sample sequentially. Secondly, although the sample stays coherent, long term structure is not observed (such as repeating chores). Thirdly, noticeable noise are introduced during the upsampling stage by different upsampler; and lastly, it has too many parameters and requires very long time to train.

4.4 Conclusion

In this chapter we discuss the various techniques in different areas of music generation. We also looked into two notable papers in each of them in details, and these techniques will be further investigated in the following section. The current SOTA in music generation still need many improvements to reach the level of what a professional music producer could do, but improvement throughout recent years bring us closer to that goal. A notable trend is that researchers are moving from generating short melody to generating polyphonic with long term structure. More recent works such as Jukebox has shown to be able to do this even in the raw audio domain, which is very promising.

Chapter 5

Experiments

5.1 Approach

In this chapter, I will be introducing the experiments I have done. Throughout the entire year, I have spent a significant amount of time learning and experimenting with many different techniques. You can find all experiments I have descibed in the submitted code. In this report, I will focus on describing the attention-based experiments that I have done.

I will first gives an overview of the libraries I had used and evaluation approach in later subsection, then I will discuss the implementation and compare the performance of traditional RNN and it's simple attention variant; residual connection and embedding mechanism will also be discussed and compared along side. These models will be evaluated under both monophonic and polyphonic generation tasks, and the details is described in the monophonic experiment section and polyphonic experiment section respectively.

The attention experiment section will describe the implementation of various architecture using the attention mechanism, and investigate the effects of varying the number of heads (of multihead attention) and stacks (of part of the transformer architecture). I will also introduce pre-activation and xavier uniform to improve the stability of training process.

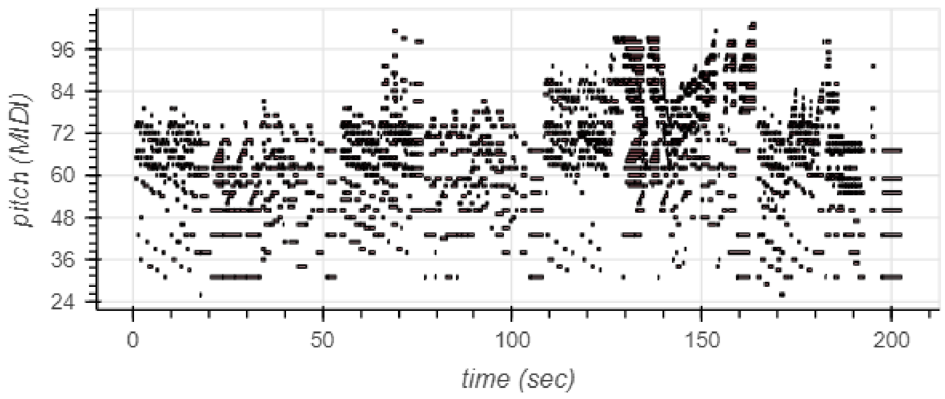
Lastly in the failed experiment section, I will describe two failed attempts in the early stage of the project using convolution-based mechanism, which I have learned most and built a solid foundation for my later experiments.

5.1.1 Libraries

In this subsection, I will briefly introduce the libraries that I have used. Since the project is written using python[1], the standard library is not introduced.

- **Pytorch**[2] All of my experiments are based on the pytorch machinel learning framework.
- **Note Seq**[3] Used for preprocessing midi files in some of the experiment. Figure 5.1 shows the comparison of note seq implementation and my implementation by processing the same midi file.
- **Mido**[4] Used for building my own preprocessing toolkit, provide low level operation on midi files.

[1]`https://www.python.org/`
[2]`https://pytorch.org/`
[3]`https://github.com/magenta/note-seq`
[4]`https://github.com/mido/mido`



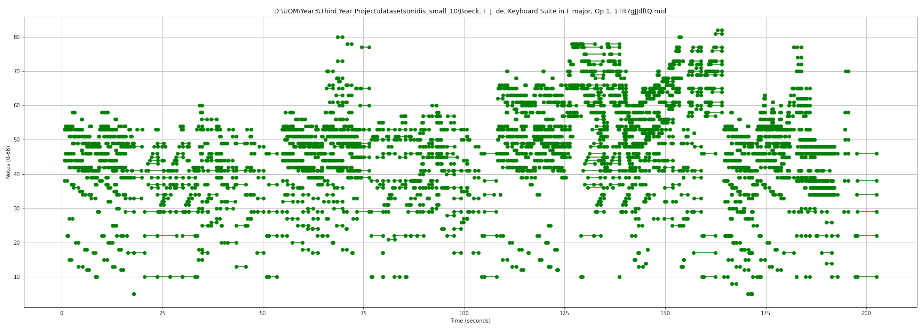


Figure 5.1: Visualization of the preprocessed note sequence of the same song. Bottom figure shows my own implementation and top shows magenta note seq's implementation. X-axis denotes the time in seconds; y-axis denotes the pitches being played; green/black lines/dots indicate the note is being played at that time.
Boeck, F. J. de, Keyboard Suite in F major, Op.1

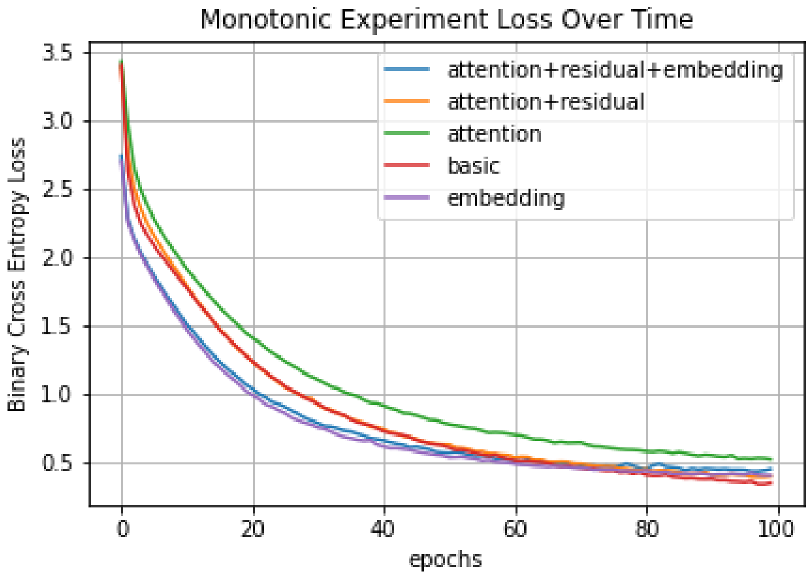


Figure 5.2: Loss over time for different models of monophonic experiment.

- **Music21**[5], similar to mido, but was not favored after using it for initial experiment.
- **Numpy**[6] and **Scipy**[7] Scientific libraries for common processing.

5.1.2 Evaluation

Figure 5.2 shows the loss over time of different models in one of my experiment. It can be observed that the attention residual embedding network and embedding network converge fastest, which suggests that the embedding layer plays an important role in these music generation models, but the loss only suggests that the embedding layer helps convergence, but gives little indication of the quality of the generated music. Indeed, in the field of music generation, the preferred judge is the subjective preference of human, but at the same time, a number of objective measurements can also be made to provide rough pointers of the performance of the system (Ji et al., 2020).

As I was not able to collect enough subjective evaluation data, I investigated on a number of objective measurements to assess the quality of music generated by the models. There are many different approaches for evaluation. Classification introduced here will be similar to those used in (Yang and Lerch, n.d.).

- **Model Metrics**. Metrics that does not based on musical theoretical knowledge, for example the precision, recall, BLEU score[8] and perplexity[9].
- **Descriptive Statistics**. Statistical data based on musical concepts, including pitch, rhythm, chord, and style transfer.

[5]`https://web.mit.edu/music21/`
[6]`https://numpy.org/`
[7]`https://www.scipy.org/`
[8]`https://en.wikipedia.org/wiki/BLEU`
[9]`Perplexity`

- **Other**. Algorithms exist to analysis other aspects, such as the temperature of tone and structure. For example, detecting repeated patterns in the generated music (Wang and Dubnov, n.d.) and signal-to-noise ratio (SNR)[10].

In this report, I will be using the descriptive statistics based on (Yang and Lerch, n.d.), and I will describe each of them below:

- **Octave Transition Histogram**: This shows the spread of octave[11]-independent pitch with dimensionality[12] of 12.
- **Pitch Transition Matrix**: This similar to pitch transition histogram, but it shows the transitions of each pair of octave instead, forming a 12×12 matrix.
- **Intra-set Distance**: This shows the difference between the occurrence of pitches using euclidean distance.
- **Inter-set Distance**: This shows the difference between the occurrence of pitches between two datasets using euclidean distance.
- **Average Inter-onset-interval**: This shows the average distance between two consecutive pitches is being played.
- **KL Divergence**: How the probability distribution of pitches differ between two datasets, similar to inter-set distance.

5.2 Settings

5.2.1 Datasets

There are many datasets used in this project, although there will be no explicit mention of dataset during the description of experiment, you can find their sources from below. This is because the actual dataset used are quite arbitrary as they are often a mixture from different sources and styles. However, one can experiment with different datasets at will.

1. The Largest MIDI Collection on the Internet [13].
2. Bhs minor9[14].
3. Maestro[15].
4. Byte Dance Giant Piano[16].

[10]https://en.wikipedia.org/wiki/Signal-to-noise_ratio

[11]https://en.wikipedia.org/wiki/Octave

[12]This refers to the chromatic quantization of the frequency continuum, in western music theory, each octave is made of 12 pitches, but they still refers to the same octave, therefore each octave is calculated as the sum of pitches belong to the same octave.

[13]https://www.reddit.com/r/WeAreTheMusicMakers/comments/3ajwe4/the_largest_midi_collection_on_the_internet/

[14]https://bhs.minor9.com/

[15]https://magenta.tensorflow.org/datasets/maestro

[16]https://github.com/bytedance/GiantMIDI-Piano, (An email request is needed)

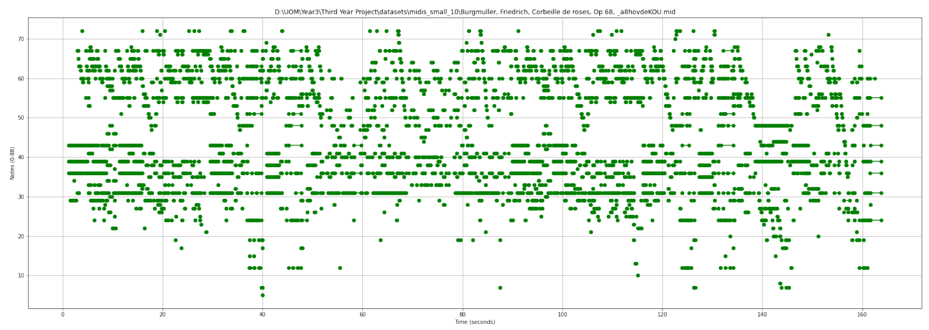


Figure 5.3: Visualization of the preprocessed midi file. X-axis denotes the time in seconds; y-axis denotes the pitches being played; green lines indicate the note is being played at that time. Burgmuller, Friedrich, Corbeille de roses, Op.68

5. Piano E Competition [17].
6. Theory tab [18].

5.2.2 Pianoroll

I started my experiment with my own implementation of midi files conversion, it utilizes a low level library called mido[19], mido is a library for working with midi messages and ports, but I will only use it to read the midi files into sequence of events, which is then converted into a pianoroll representation. For simplicity, I only considered `note on` and `note off` event at this stage, with no velocity[20] and pedal control[21]. As a result, there are slight loss of quality to the original audio, but by listening to multiple tracks that preprocessed in this way, I found that the music samples are largely similar. At the same time, I also developed my own visualizing tools to display the preprocessed pianoroll, an example of visualization is shown in figure 5.3.

5.2.3 Playing the midi

I also written a small library which utilizes Fluid Synth[22] to play a midi file in jupyter-notebook. It works by converting the midi file to a wav file with the specified soundfont path using the the synthesis functionality from Fluid Synth. Then it simply loads the temporary file onto jupyter-notebook. The `midi.utils` package also contains some convenient functions for directly playing the preprocessed pianoroll.

5.3 Monophonic Experiment

Monophonic (Melody) is a sequence of notes, with different pitch, duration and velocity. At each time step, only a single note is being played, therefore the model only need to predict the probability of single

[17] `https://www.piano-e-competition.com/`, (I have written a crawler in `monotonic_experiment/download.ipynb`)
[18] `https://www.hooktheory.com/theorytab`
[19] `https://github.com/mido/mido`
[20] The speed of hitting a note on the piano.
[21] The item control by the foot of the pianoist to make modification to the current playing music
[22] `https://github.com/FluidSynth/fluidsynth`

note to be played in the next time step.

5.3.1 Representation

The midi files are first converted into the form of pianoroll (a matrix of pitch $\times$ time-step, see sec. 5.2.2). For each timestep, the current active note is encoded, if there is multiple active note, the one with highest pitch is used. The sequence of note is also quantized to 120 quarters/beats per minute, and each quarter is set to 4 ticks. According to the midi specification [23] each midi event is encoded in 8 bits, and 1 extract bit to encode silence event, therefore in total we have a vector of size 129 at each time step.

Models

I implemented and experimented with 5 types of models:

- **Vanilla (Basic) RNN**: The one-hot encoded input is first passed to a stack of 2 layer RNN using LSTM cells, where each LSTM cell consists of 256 hidden units and a dropout layer with probability 0.5; follow by a dense output layer that maps the 256 hidden units to 129 output classes.
- **Attention RNN**: A self-attention layer with projection is added before input to the second LSTM layer to allow the model to attend to different region of the LSTM model.
- **Attention Residual RNN**: In addition to Attention RNN, a residual connection is added around the attention output in attempt to enhance gradient flow.
- **Attention Residual Embedding RNN**: An embedding layer is used to capture semantic meaning of the input pitch on top of Attention Residual RNN.
- **Embedding RNN**: An embedding layer is added to the Basic RNN model.

Representation

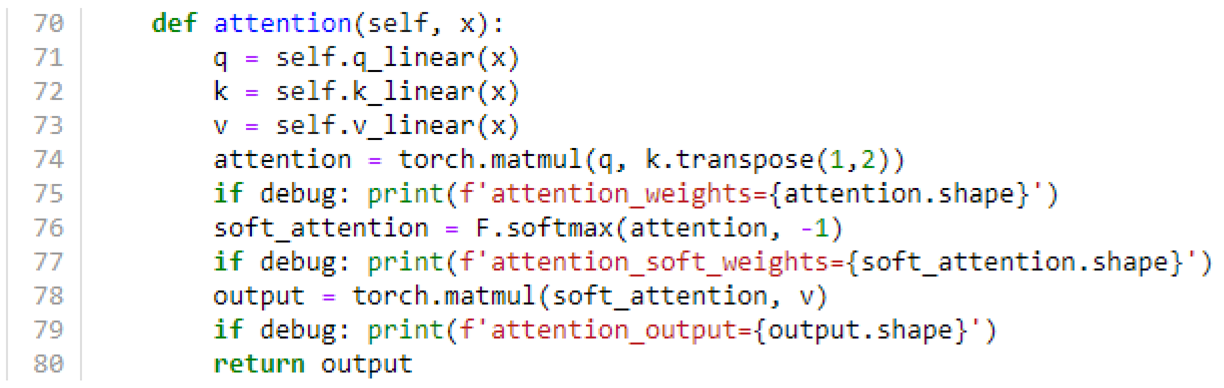
Since only one note is being played at any specific time, we can encode the sequence of pianoroll as a sequence of number, where repetitions of the same number denote the duration of pressing the same key. The input sequence is also quantized (see sec. 3.2.9) with unit of 0.01. In addition, for each model if embedding is not used, the input sequence is also converted to its one-hot representation.

5.3.2 Implementation

There are many variants of the attention (see sec. 3.2.3) mechanisms, for example, the original additive attention (Bahdanau et al., 2016); the dot product attention (Luong et al., 2015); and the scaled dot product attention (Vaswani et al., 2017), I am using dot product self attention with projection, the code is shown in figure 5.4.

For all five models, Residual connection (see sec. 3.3) is applied to the attention module, where the output from the attention is added to the input of the attention. Dropout with probability 0.4 is applied at the end of each RNN layer. The embedding dimension is set to 50. Gradient clipping (See sec. 3.2.13) is applied and the maximum gradient is set to 3.0. The hidden size of both LSTM is set to 256. At the end of the last RNN layer, a linear layer is used to map the last output of the LSTM to number of classes.

[23] `https://www.midi.org/specifications/file-format-specifications/standard-midi-files/rp-001-v1-0-standard-midi-files-specification-96-1-4-pdf`



```
70      def attention(self, x):
71          q = self.q_linear(x)
72          k = self.k_linear(x)
73          v = self.v_linear(x)
74          attention = torch.matmul(q, k.transpose(1,2))
75          if debug: print(f'attention_weights={attention.shape}')
76          soft_attention = F.softmax(attention, -1)
77          if debug: print(f'attention_soft_weights={soft_attention.shape}')
78          output = torch.matmul(soft_attention, v)
79          if debug: print(f'attention_output={output.shape}')
80          return output
```

Figure 5.4: The implementation of the self attention. `q/k/v_linear` denotes the linear projection of the input; line 74 computes the weighting for value; line 76 computes the score from the weighting using the softmax function; line 78 computes the output using the calculated score and the projected input.

The Adam (Kingma and Ba, 2017) optimizer is used with learning rate of 0.001. Through some experiment I also find that the lr scheduler[24] does not work well and thus none was used.

With combination of downloading and crawling the web, I gathered a number of datasets, a subset was used and it consists of about 45k training samples in total. All five models are trained for 100 epochs. Non-cherry picked samples can be found in the following urls:

- Attention Residual Embedding RNN: `https://bit.ly/mono_attn_res_emb_rnn`
- Attention Residual RNN: `https://bit.ly/mono_attn_res_rnn`
- Attention RNN: `https://bit.ly/mono_attn_rnn`
- Basic RNN: `https://bit.ly/mono_rnn`
- Embedding RNN: `http://bit.ly/mono_emb_rnn`

5.3.3 Evaluation

For each trained model, I generated 500 one minute long samples, and the evaluation is done between these datasets.

Figure 5.5 shows the transition matrix of different models. From the real matrix, we can see that it has strong tendency of transitioning to the same octave, while the all models tends to shift one octave up. This is expected as the monophonic representation capture the higher octave if multiple pitches are being played at the same time. We can also observe that there are more similarity on the overall structure on the right and bottom side, this suggests that the models are generally better on capturing the transition of higher octaves than lower octaves. If we take a closer look at their differences by calculating the KL-divergence of the values in the transition matrix shown in table 5.1, we can see that the attention residual embedding has the closest distribution, followed by attention residual, embedding, attention and basic model.

Figure 5.6 shows the distribution of distances of intra and inter-set between generated data and the real dataset, it shows that all models performs expectedly as the regions with more pitches being played is getting more error as well, however, the hip between pitch 10 and 20 was not well captured, this is

[24]A technique that varies the learning rate throughout the training process for faster convergence.

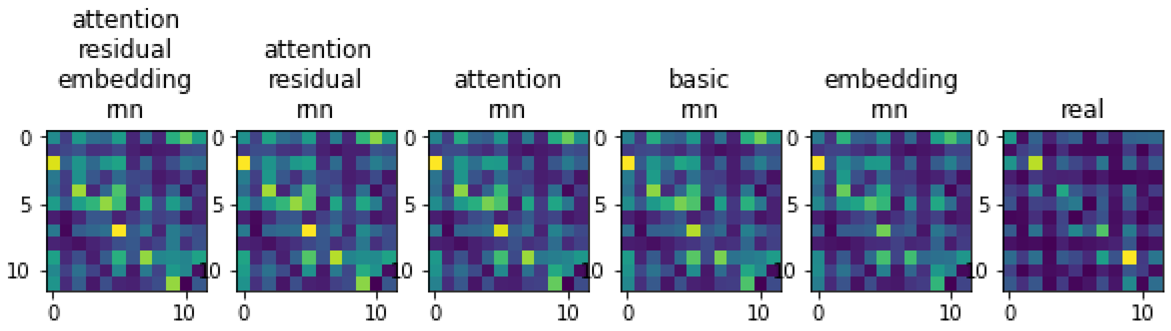


Figure 5.5: Monotonic experiment: 12 by 12 Octave Transition Matrix of different models

Monotonic experiment: KL-divergence of transitions ocurrences

model	KL-divergence
attn res emb	0.0431
attn res	0.0482
attn	0.0495
emb	0.0485
basic	0.0502

Table 5.1

probably due to the real dataset had been converted to monophonic sequence and the pitches at that area is not present during training.

Table 5.2 and 5.3 shows the corresponding sum of distances. By observing the inter-set distances in the table, we can find that the embedding rnn model is closest to the real dataset, followed by basic, attention, attention residual and attention residual connection; however, the ranking inversed if we were to look at the intra-set distances, where the attention residual embedding rnn has the closest distribution with the real dataset. Combining the sharp error distribution shown in figure 5.6, we can make reasonable suggestion that the basic rnn is probably overfitting to part of the data in the real dataset, and embedding helped to regularize the model.

In conclusion, the models with attention mechanism shown to be able to model the distribution of the pitches of the real dataset better than vanilla rnn, and with the help of residual connection and embedding layer, the model can be further improved. At the same time, embedding layer and the attention mechanism are also seems to have some regularization capability to prevent overfitting which is plausible.

Monotonic experiment: sum of intra distances

model	sum distance
attn res emb	1469.066
attn res	1507.901
attn	1526.397
emb	1583.879
basic	1553.075

Table 5.2

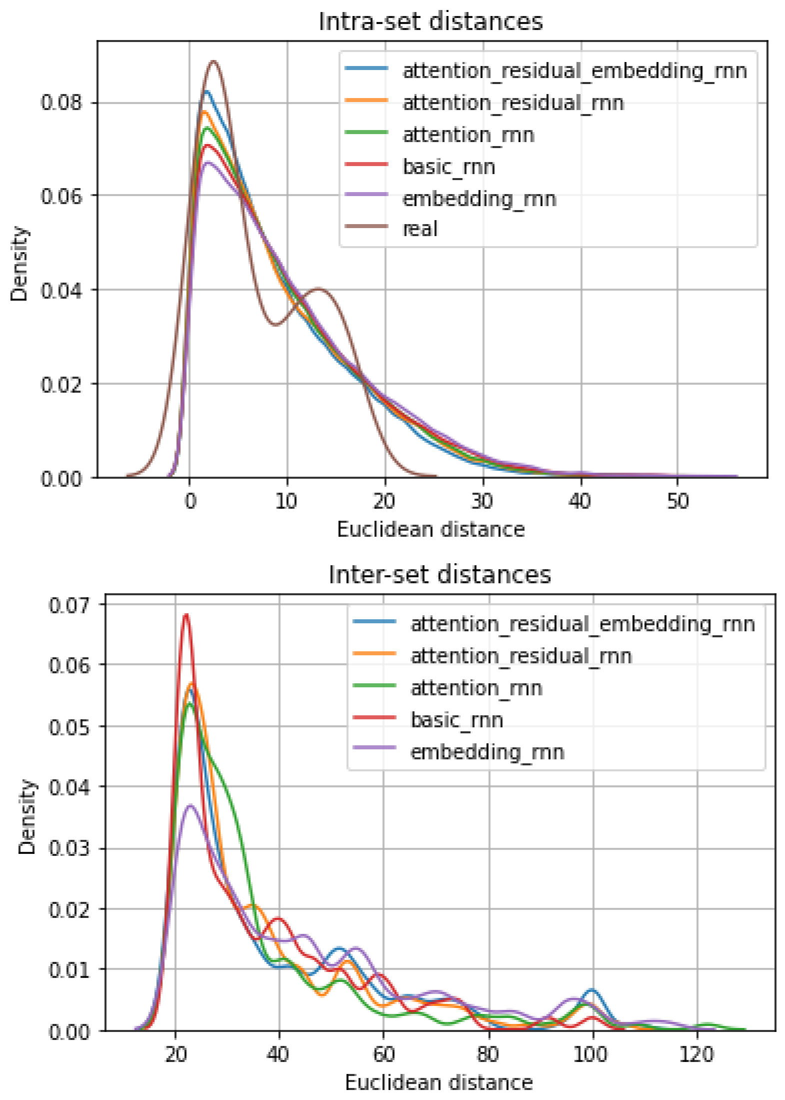


Figure 5.6: Monotonic experiment: Kernel density estimation (this models the distribution of the data in the dataset, similar to histogram) of Intra-set (top) and inter-set (bottom) distance of different models.

Monotonic experiment: sum of inter distances

model	sum distance
attn res emb	3255.256
attn res	3142.493
attn	3034.470
emb	2949.661
basic	2981.271

Table 5.3

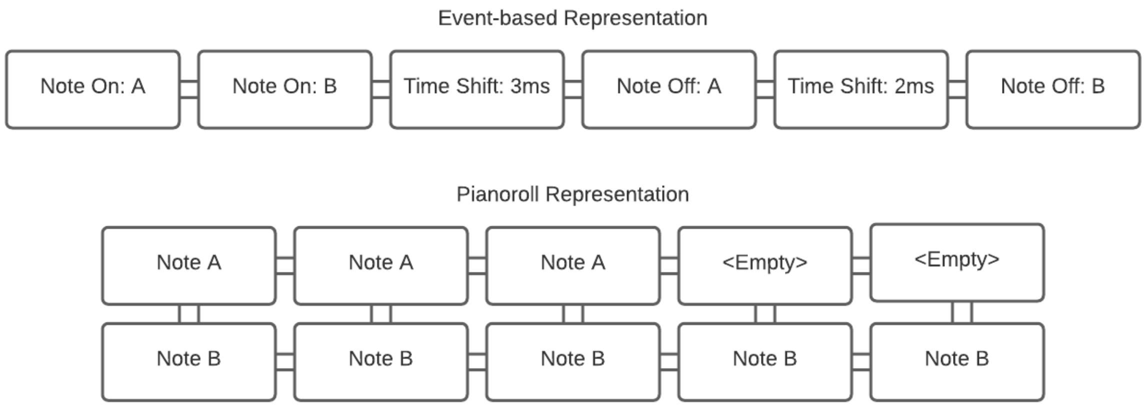


Figure 5.7: Example of playing Note A for 3ms and playing Note B for 5ms simultaneously in the event representation (top) and pianoroll representation (bottom). Here pianoroll assumed duration for each block is 1ms.

5.4 Polyphonic Experiment

Polyphonic refers to multiple notes can be played at the same time, and therefore it requires a more complicated encoding scheme. In this section I will discuss some of the experiments I have done.

5.4.1 Representation

I attempted two representations in my experiments, pianoroll from (Dong et al., n.d.) and event-based from (Roberts et al., 2019).

Pianoroll

Pianoroll was expressed as a multi-class classification problem, where the input to the model is a sequence of vectors, each consists of 88[25] entries, and its value is 1 if the key is being played at that time-step and 0 otherwise. Similar to the monophonic task, the duration of key press is encoded as continuous 1s in the sequence.

Event

Event-based representation encode simultaneous key press in a sequential manner; and time information is encoded as time-shift event instead of continuous occurrence of the same events. In (Roberts et al., 2019)'s implementation, four different type of events are used: *note on* encode start of an key being pressed, *note off* encode end of an key being pressed, *time-shift* encode time elapsing and *velocity* encode how hard has the performer pressed the key.

Figure 5.7 further demonstrate the difference between event and pianoroll representation.

[25]This is the number of keys in the piano

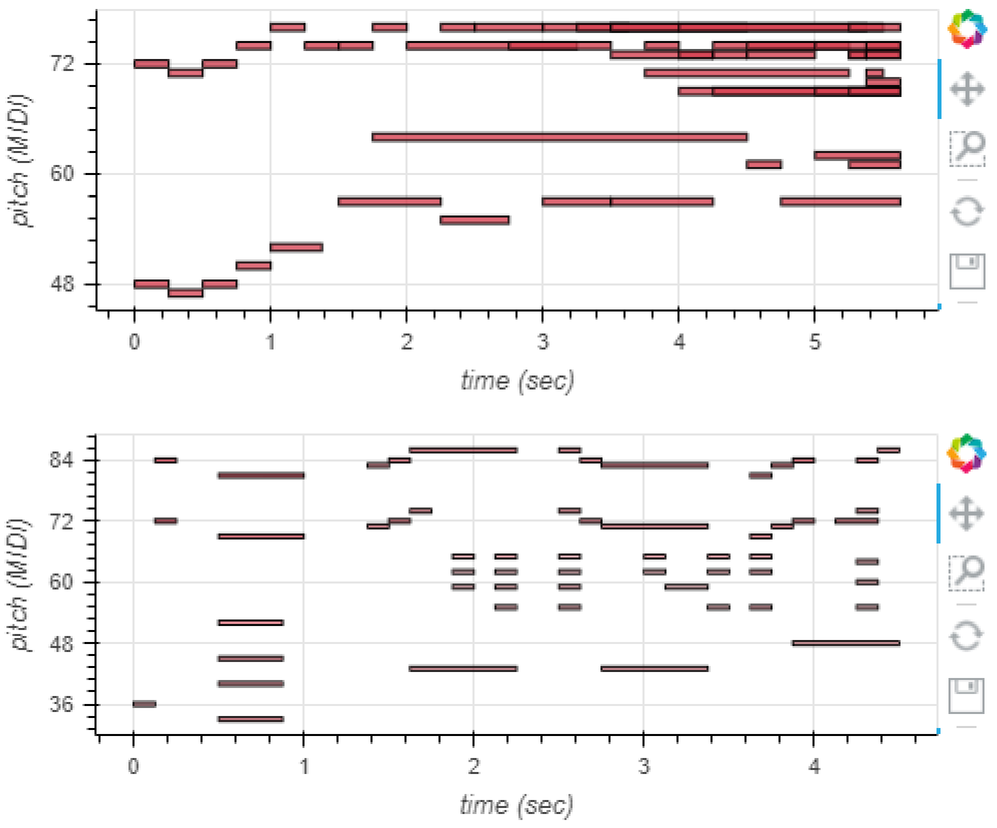


Figure 5.8: Polyphonic experiment: example of a deteriorating sample's pianoroll (top) and a pianoroll from the real dataset (bottom).

5.4.2 Implementation

The same five models are used for training but with different configuration. Since polyphonic sequences are harder to model, the dropout probability is lowered to 0.2. I also used an subset of the Piano E Competition dataset (see sec. 5.2.1) as it is useful for learning dynamic with notes (*Performance RNN: Generating Music with Expressive Timing and Dynamics*, n.d.), in total there were around 175k training samples and each model is trained for 100 epochs. Non-cherry picked samples can be found in the following urls:

- Attention Residual Embedding RNN: `https://bit.ly/poly_attn_res_emb_rnn`
- Attention Residual RNN: `https://bit.ly/poly_attn_res_rnn`
- Attention RNN: `https://bit.ly/poly_attn_rnn`
- Basic RNN: `https://bit.ly/poly_rnn`
- Embedding RNN: `http://bit.ly/poly_emb_rnn`

5.4.3 Evaluation

The following evaluation is based on 200 generated samples of around 1 minute for each model. While generating some initial samples, I noticed that in some cases, the quality of audio deteriorate fairly

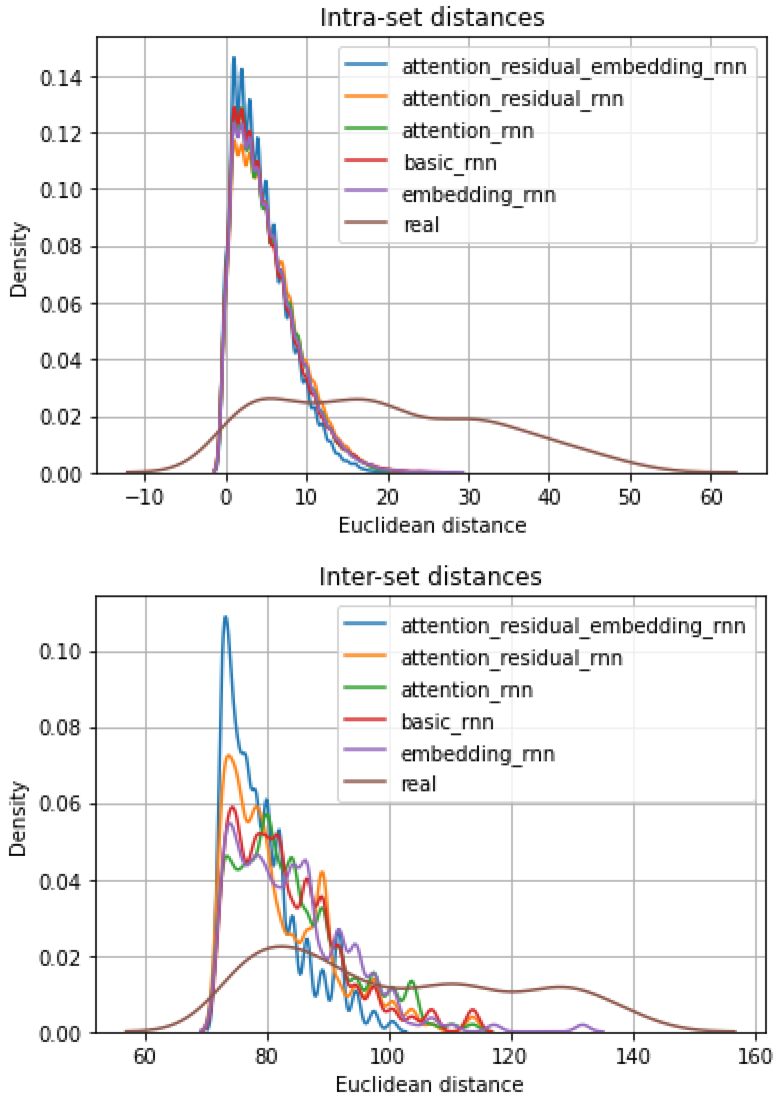


Figure 5.9: Polyphonic experiment: Kernel density estimation (this models the distribution of the data in the dataset, similar to histogram) of Intra-set (top) and inter-set (bottom) distance of different models.

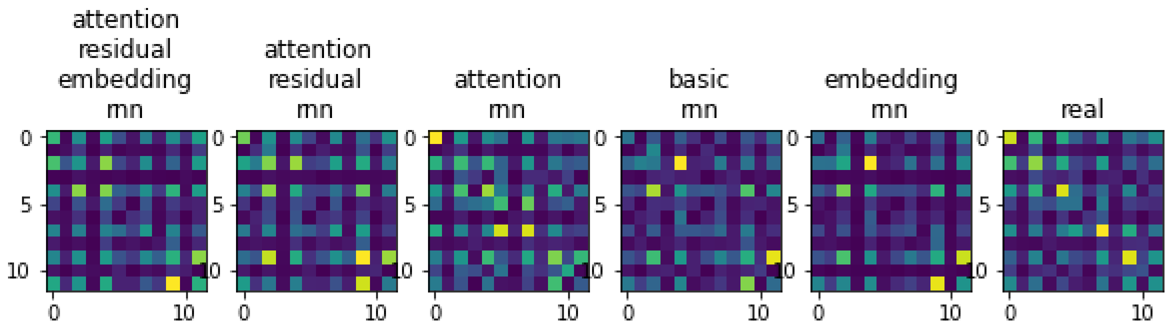


Figure 5.10: Polyphonic experiment: 12 by 12 Octave Transition Matrix of different models

Polyphonic experiment: sum of inter distances

model	sum of distances
attn res emb	1781.840
attn res	1813.039
attn	1834.427
emb	1834.581
basic	1826.264

Table 5.4

quickly, a comparison is shown in figure 5.8. We can observed from the deterioating sample that towards the end of the pianoroll at the top right corner, some notes are being played in a chaotic way. By observing more samples, I found that the model often forgets to turn off the turned on-ed notes, leading to some notes being played for a very long time. Since the length of the precondition is 128 in the experiment, and *note on* and *note off* events are used to denote the duration of notes, it will lead to a phenomenon where if a note is turned on for more than 128 events before turning off, the model will have no information about the note being turned on and therefore will not attempt to turn it off again. By observing figure 5.9, we can find these pitches that got affected by this phenomenon, since those notes never get turned off, they will have much higher occurrences, denoted by the sharp spikes in the intra-set distance density diagram. We can see that a large number of pitches are affected by it, in particular, from the inter-set distanes, we can see that the attention residual embedding layer is affected by this problem the most, denoted by the sharp spike in the graph.

Figure 5.10 shows the octave transition matrix of different models, we can observed that the overall structure of these matrices are closer to those in the monophonic experiment. This is because the harmonies/chord information is preserved.

Another observation from figure 5.10 is that although there are relatively even distribution of the pitches in the real data (can be seen from the flat curve of the real data in the intra-set distance), we can see that the error of models still tends to focus on the lower pitch regions. Interestingly, this phenomenon is also briefly described in the jukebox experiment (see sec. 4.3.2), where they applied short-time fourier transform and attempt to match the distribution.

By observing table 5.4 and 5.5 we can also observe that the performance of the attention models and basic model are similar, it is only when the embedding and residual is introduce the attention models starts to perform better than the basic rnn. The average inter-onset-interval (see sec. 5.1.2) is also calculated, shown in table 5.6, and the performance of all models are similar and are able to match the frequency of the real dataset.

Polyphonic experiment: sum of intra distances

model	sum of distances
attn res emb	423.424
attn res	454.790
attn	444.436
emb	447.900
basic	442.526

Table 5.5

Polyphonic experiment: average inter-onset-interval

model	interval (time-step)
attn res emb	0.104
attn res	0.106
attn	0.091
emb	0.099
basic	0.108
real	0.114

Table 5.6

In conclusion, all of the models yielded similar performance in this experiments, but combining residual connection and embedding, the model can have significant improvement over the vanilla rnn. However they all suffer a similar issue where the model has problem remembering notes that play for a long time, and we also find that most error occurred in the lower pitch region, which may be worth to investigate further.

5.5 Attention-based Experiment

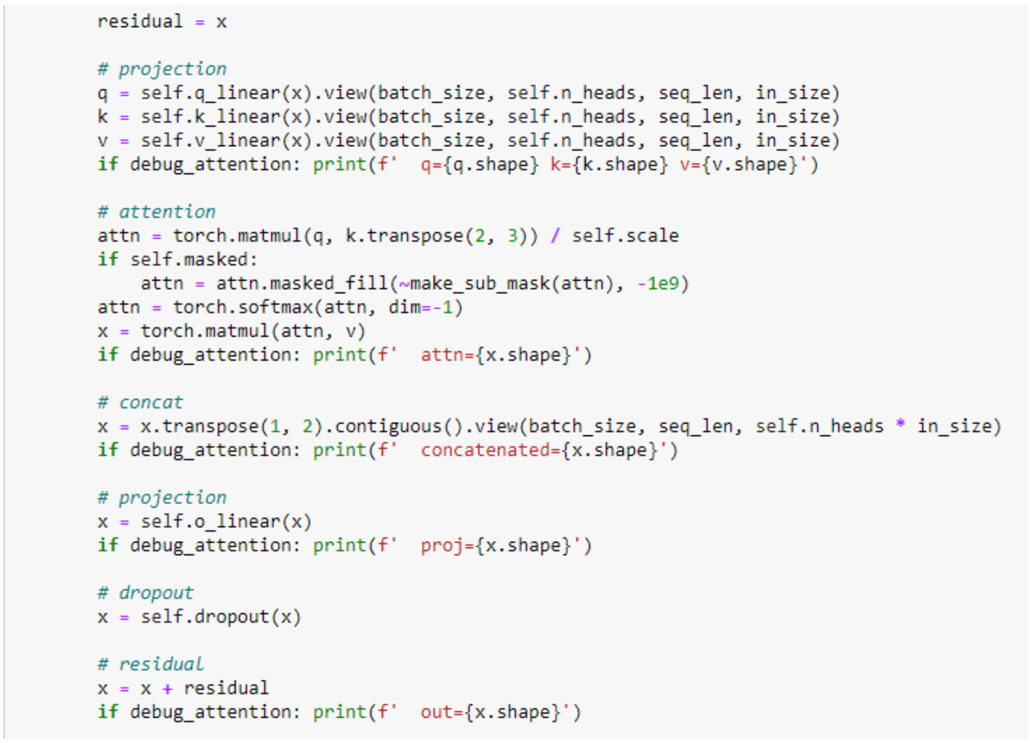
In this section, I will describe an experiment that I have done to investigate the use of the attention mechanism in music generation. Note that the model architecture here is purely for experiment purpose and they have limited capacity in order to enhance the differences between them because some early experiment using a large model shown that it can be difficult to compare and analysis the properties of generated music. However, one can easily train a model with larger capacity and feed in more data for a better result.

5.5.1 Representation

The representation used in this experiment is same as the one used in polyphonic experiment, therefore I preferred not to reiterate it again here (see sec. 5.4).

5.5.2 Implementation

In this experiment, the attention mechanism through a sub-module of the experiment, I have implemented both the encode and decoder in the transformer (Vaswani et al., 2017) architecture, but instead of the transformer architecture, I will use only the decoder in this experiment.



```
        residual = x

        # projection
        q = self.q_linear(x).view(batch_size, self.n_heads, seq_len, in_size)
        k = self.k_linear(x).view(batch_size, self.n_heads, seq_len, in_size)
        v = self.v_linear(x).view(batch_size, self.n_heads, seq_len, in_size)
        if debug_attention: print(f'  q={q.shape} k={k.shape} v={v.shape}')

        # attention
        attn = torch.matmul(q, k.transpose(2, 3)) / self.scale
        if self.masked:
            attn = attn.masked_fill(~make_sub_mask(attn), -1e9)
        attn = torch.softmax(attn, dim=-1)
        x = torch.matmul(attn, v)
        if debug_attention: print(f'  attn={x.shape}')

        # concat
        x = x.transpose(1, 2).contiguous().view(batch_size, seq_len, self.n_heads * in_size)
        if debug_attention: print(f'  concatenated={x.shape}')

        # projection
        x = self.o_linear(x)
        if debug_attention: print(f'  proj={x.shape}')

        # dropout
        x = self.dropout(x)

        # residual
        x = x + residual
        if debug_attention: print(f'  out={x.shape}')
```

Figure 5.11: Code snippet of the implementation of multihead attention.

The multihead attention's (see sec. 3.2.3) implementation is quite similar to the usual attention with the following addition according to the original paper (Vaswani et al., 2017). Firstly, the input is projected as usual, but each head has individual projection; next, the scaled dot product self_attention is applied (see sec. 3.2.6); then the output from different attention heads are concatenated together and a final projection is applied to get the desired output size. Figure 5.11 shows the detailed implementation.

I also made two modifications that shown improvements in the initial experiments. First is the use of pre-activation (He et al., 2015) in the multihead attention, which means the activation is used before the operations in the residual network and secondly to initialize the weight with xvaier uniform as described in [26], which keeps the value of weight uniform so that the weights can be reached into deeper network. Figure 5.12 shows the overall architecture of the network.

I used the following architectures for the experiment:

1. 1 stack of decoder, 1 head for each multihead attention.
2. 1 stack of decoder, 3 heads for each multihead attention.
3. 3 stacks of decoder, 1 head for each multihead attention.
4. 2 stacks of decoder, 2 heads for each multihead attention.

where stack of decoder refers to using two layers of decoder where the output from the last layer is passed to the next.

Dropout with probability 0.4 is applied to the end of each multihead attention and feed forward module, and the output of the decoder network is feed into two layers of linear to output the number of classes.

[26]glorotUnderstandingDifficultyTraining

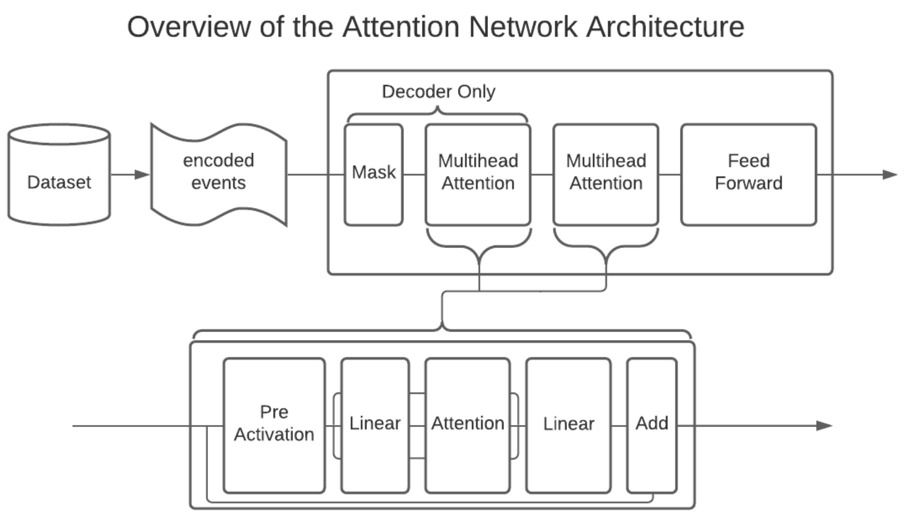


Figure 5.12: Overview of the attention network architecture.

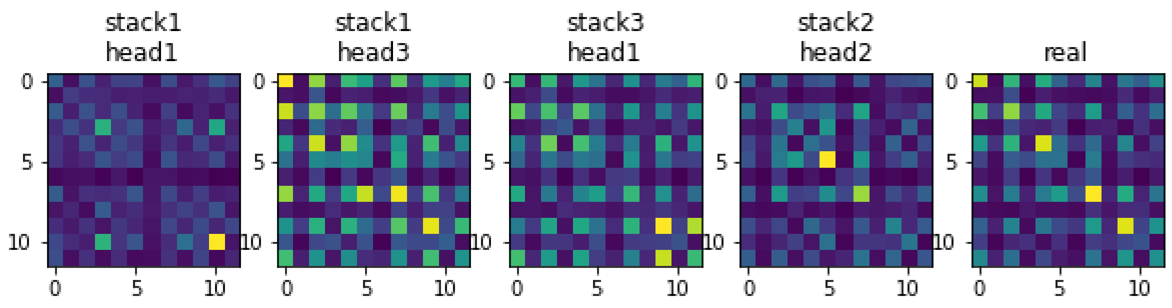


Figure 5.13: Attention experiment: 12 by 12 Octave Transition Matrix of different models

A subset of the large dataset (see sec. 5.2.1) was used, collectively made up of around 40k training samples, similar to previous experiments, each model is trained with 100 epochs. Since the experiment is mainly to find the working of attention mechanism, the samples are generally not in good quality, but there are also non-cherry picked samples in the following urls:

- **stack 1 head 1**: `http://bit.ly/stack1_head1`
- **stack 1 head 3**: `http://bit.ly/stack1_head3`
- **stack 2 head 2**: `http://bit.ly/stack2_head2`
- **stack 3 head 1**: `http://bit.ly/stack3_head1`

5.5.3 Evaluation

Figure 5.13 shows the octave transition matrix of multiple models. We can observe that the model with 1 stack and single head attention do not have the capacity to model a plausible distribution of the real dataset; the 1 stack 3 attention heads models seems to best model the real distribution, followed by 3

Attention experiment: sum of inter distances

model	sum distance
stack 1 head 1	2088.857
stack 1 head 3	1775.623
stack 3 head 1	1860.495
stack 2 head 2	1860.365

Table 5.7

Attention experiment: Mean of the used pitches

model	mean
stack 1 head 1	48.98
stack 1 head 3	35.505
stack 3 head 1	37.285
stack 2 head 2	37.325
real	31.4

Table 5.8

Attention experiment: euclidean distance to the real dataset

model	raw pitch	transposed pitch
stack 1 head 1	2088.857	54.622
stack 1 head 3	1775.623	56.807
stack 3 head 1	1860.495	58.191
stack 2 head 2	1860.365	57.307

Table 5.9

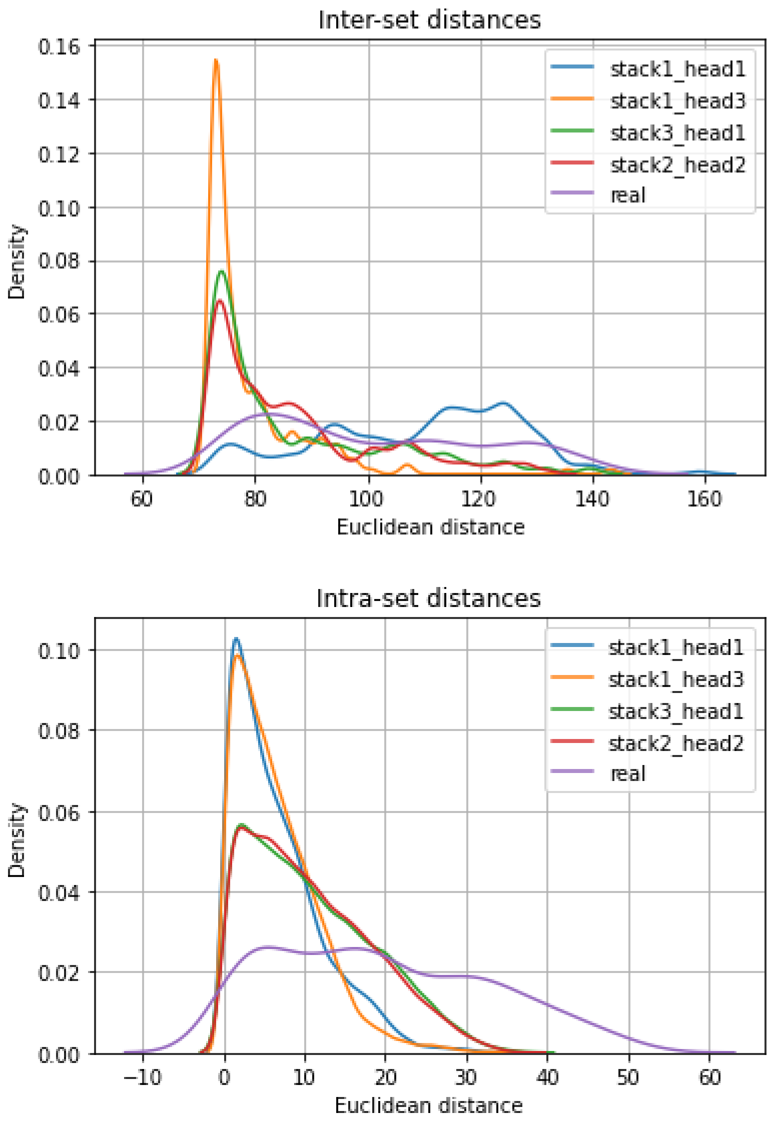


Figure 5.14: Attention experiment: Kernel density estimation of Inter-set (top) and intra-set (bottom) distance of different models.

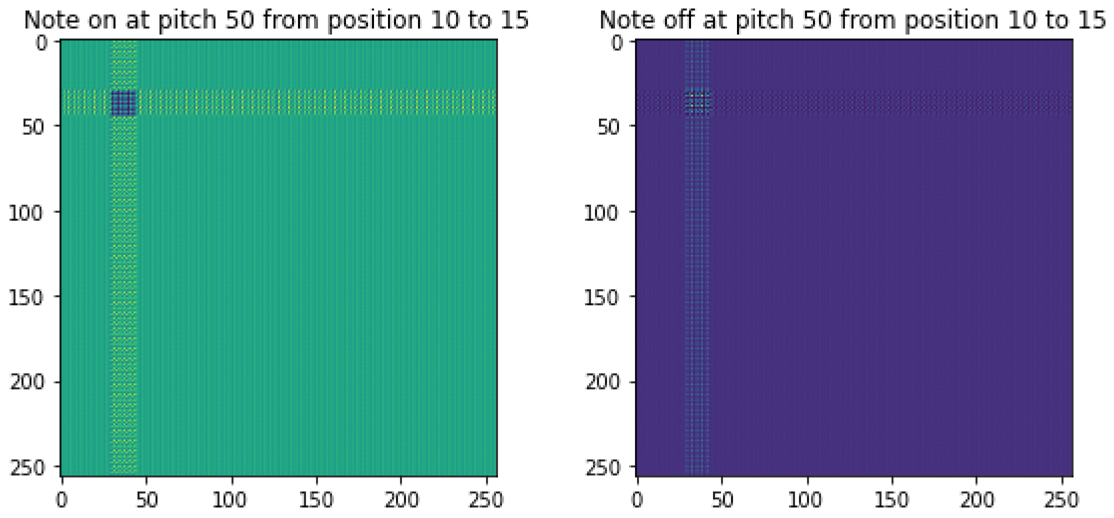


Figure 5.15: The attention score image of note on (left) event and note off (right) at pitch 50, from sequence position 10 to 15. Bright area indicate higher score.

stack 1 attention head and 2 stack 2 attention heads. Table 5.7 and 5.8 showing the inter-set distances and used pitches also further strength this hypothesis.

A sharp increase can be observed from the inter-set distance of 1 stack 3 heads from figure 5.14, and this may be a sign of overfitting, but we can find that the distribution of the model remain a reasonable distribution by observing the intra-set distribution; and the euclidean distance of raw and transposed pitches from table 5.9 also matches, which shows that the model is performing well on all octaves not limiting to only a subset of the data. These evaluation shows that the multihead attention is most important factor in the attention mechanism.

Attention

We can visualize the attention by tweaking its input. Figure 5.15 shows computed score for note on and note off event, we can see that the network shows an almost inverted score for these two events, which made sense since these two events are completely oppose of each other.

We can also visualize the attention score by putting the note on event and note off event on the same input. Figure 5.16 shows how two events interact with each other on the same attention score image, we can see that these two events from a square where the four corners has most attention, it seems like the note on event learns to find the corresponding note off event by assigning more score to region in four directions (top, bottom, left, right), which is interesting.

I also notice in the first attention layer, the model tends to capture more of the absolute positional information, shown in figure 5.17 and more periodic data in the later layer, shown in figure 5.18, which is interesting as well.

In conclusion, this experiment shows that multihead attention is crucial for the attention-based model, more heads are preferred over more layers when given the same number of parameters. Visualizations also shows the inner working of the attention mechanism, giving some insights of how it works internally.

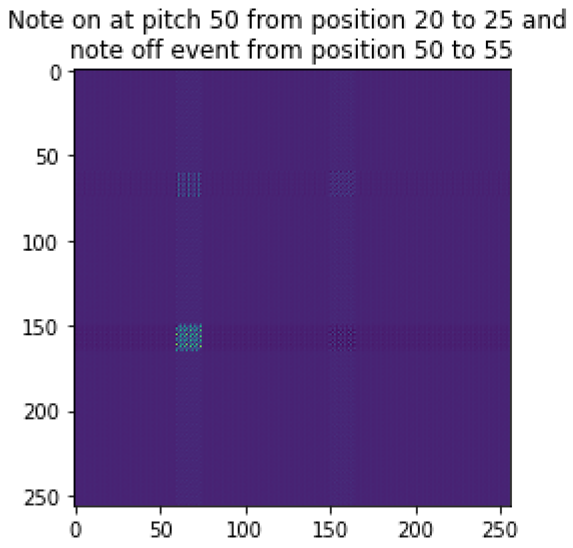


Figure 5.16: The attention score image of note on event and note off at pitch 50, from sequence position 10 to 15 and 50 to 55 respectively. Bright area indicate higher score.

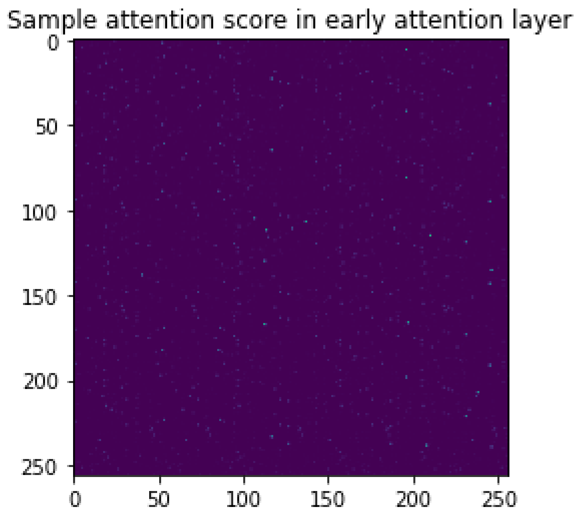


Figure 5.17: Sample score of early attention layer.

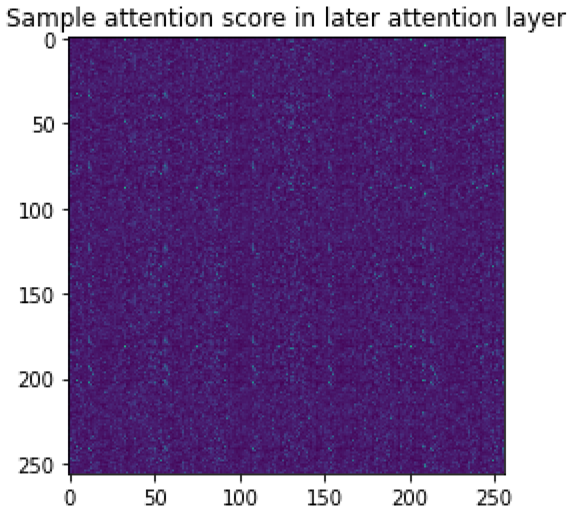


Figure 5.18: Sample score of later attention layer.

5.6 Failed Experiment

This section will briefly describe two failed experiments that I have done, base on polyphonic music generation with pianoroll representation.

5.6.1 MidiNet-like GAN

This is one of earliest attempt, although the implementation is correct (validated with author code), but the representation of the data was too fine-grained and resulted in the model unable to capture the overall representation.

Figure 5.19 shows the overview of the midinet-like generator architecture; and the discriminator is simply three convolution with large kernel size (88, 24, 16) and stride (1, 24, 16) followed by two linear layers. Similar to the original paper, feature matching and one-sided label smoothing are applied to regularize the model. The model failed to converge after long time of training (over 1000 epochs), and no plausible music is generated. This is partly due to the lengthy input (primer with length 2048, mostly repeating pattern due to lack of quantization) and many other decisions including bad design of the discriminator and stacking too many linear layers. Since this problem was found at late stage of the project, no attempt were made to experiment with the fixed design.

5.6.2 Dilated Convolution

I also attempted to explore the possibility of using dilated convolution only for music generation, there are similar attempts (angsten, 2021) and the result seems promising. The idea is similar to the original paper, first reshape the input to 1d, then apply dilated convolutions repeatedly to reduce the input dimensionality, and finally applied linear and sigmoid layer to produce the output. Dilated convolution performs better than midinet-like model as it converge pretty quickly, but the unquantized sequence put lots of pressure on the model, resulting in it not learning anything, this was shown in one of the analysis done at later experiments, where approximately 97% of the training data consists of replicating the last vector in the input sequence. At the end, the model was only able to generate music through overfitting the training data.

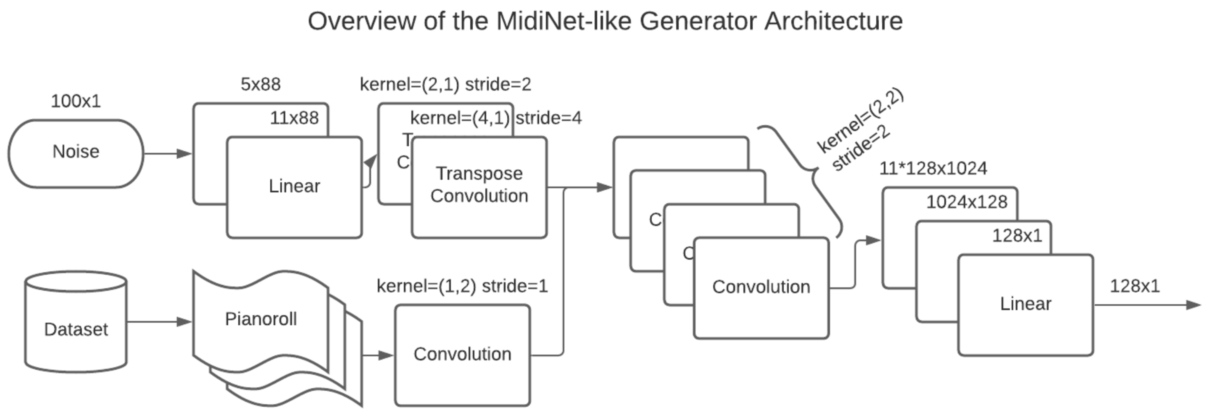


Figure 5.19: Overview of the midinet-like generator architecture. Batch normalization and relu activation is also applied after linear, convolution and transpose convolution operation. Sigmoid is applied to the output.

While implementing the network, I also spotted potential improvement on the architecture, in (angsten, 2021)'s implementation, a small number of channels were used in the early stage of the dilated convolution, but I find this caused a bottleneck and limits the performance of the model. So instead of more convolutions with small amount of channels, I increased the number of channels and used less convolutions, and observed a faster convergence.

5.6.3 Conclusion

Although some failed experiments were described in this section, they are only a small subset, in total I have done around 10 experiments that I considered failure in the early stage of the project, for example, using wavenet's residual block and pure convolutions for music generation. One can find more details in the submitted code. However, I was able to gain many helpful intuition and pointers through those failing experiments, therefore I do not regret them, and I will definitely return to them during my leisure time.

Chapter 6

Conclusion

This report presents a variety of techniques in different sub-fields of music generation, and experiments were done to analysis and evaluate one of the core mechanism behind the recent SOTA work on polyphonic music generation in the sub-field of performance generation and audio generation. However, I am only scratching the surface here, it is still an active research area and much details are yet to explore.

The title of the project Neural Music generation is a board term, there are a large amount work put into this field in the recent year, it is not trivial to have a deep understanding of the area. A large amount of time was dedicated to understanding and implementing various techniques. However, through reading and investigating the mechanism behind a number of papers, I was able to gain many knowledge not limited to music generation, but also in the boarder aspect of deep learning, which is beneficial and crucial for my future studies.

In term of planning, I felt the overall project went smoothly, although there are some delays due to my misconception, I was able to complete the project in time as I have reserved flexibility in the initial planning, partly due to the feedback from Dr. TingTing Mu which I am very grateful for.

In my opinion, the main achievement of the project would be diving and learning about different sub-fields and techniques of music generation; much time was also spent at implementing various techniques and model, it was interesting to see how machine can generate plausible composition through carefully crafted architecture. Through investigating the evaluation of music generation, we can also observe the difference of models statistically and theoretically. The failed experiments also taught me many important details in music generation, such as small changes to the representation and quantization can have huge impact on the performance of models.

Throughout the experiments, there are many interesting observations that worth to investigate further. For example, similar to (Dhariwal et al., 2020), there are much losses focus on the lower pitches, although large portions of pitches in the music pieces are low, but it should not have such a obvious spike; secondly, although we have shown that the multihead attention works better than simply stacking more single head attention, but the reason behind this phenomenon is not fully understood; lastly, the investigation towards convolution-based method was unsuccessful due to my ignorance in the early stage of the project, and it may be worth to compare the performance of convolution-based methods and attention-based methods.

In conclusion, this project had helped me gaining insights to neural music generation, as well as many deep learning techniques and knowledge, this encourage me to step deeper into the field of deep learning and provided necessary foundation for my future studies. The development of neural music generation in recent years is rapid, and I hope that one day, machine can generate music that sounds realistic and creative even compared with professional music composer.

Bibliography

Anantrasirichai, N. and Bull, D. (2021), ‘Artificial Intelligence in the Creative Industries: A Review’, *arXiv:2007.12391 [cs]* .

angsten (2021), ‘Angsten/pianonet’.

Ba, J. L., Kiros, J. R. and Hinton, G. E. (2016), ‘Layer Normalization’, *arXiv:1607.06450 [cs, stat]* .

Bahdanau, D., Cho, K. and Bengio, Y. (2016), ‘Neural Machine Translation by Jointly Learning to Align and Translate’, *arXiv:1409.0473 [cs, stat]* .

Bharucha, J. J. and Todd, P. M. (1989), ‘Modeling the Perception of Tonal Structure with Neural Nets’, *Computer Music Journal* **13**(4), 44–53.

Bretan, M., Weinberg, G. and Heck, L. (2016), ‘A Unit Selection Methodology for Music Generation Using Deep Neural Networks’, *arXiv:1612.03789 [cs]* .

Brokish, C. W. and Lewis, M. (n.d.), ‘A-Law and mu-Law Companding Implementations Using the TMS320C54x’, p. 36.

Carnovalini, F. and Rodà, A. (2020), ‘Computational creativity and music generation systems: An introduction to the state of the art.’, *Frontiers Artif. Intell.* **3**, 14.

Child, R., Gray, S., Radford, A. and Sutskever, I. (2019), ‘Generating Long Sequences with Sparse Transformers’, *arXiv:1904.10509 [cs, stat]* .

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. and Bengio, Y. (2014), ‘Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation’, *arXiv:1406.1078 [cs, stat]* .

CVAE Piano Performance Rendering - Demo (n.d.), https://sites.google.com/view/cvae-piano-perf-rend.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K. and Fei-Fei, L. (2009), Imagenet: A large-scale hierarchical image database, *in* ‘2009 IEEE Conference on Computer Vision and Pattern Recognition’, Ieee, pp. 248–255.

Dhariwal, P., Jun, H., Payne, C., Kim, J. W., Radford, A. and Sutskever, I. (2020), ‘Jukebox: A Generative Model for Music’, *arXiv:2005.00341 [cs, eess, stat]* .

Dieleman, S., van den Oord, A. and Simonyan, K. (2018), ‘The challenge of realistic music generation: Modelling raw audio at scale’, *arXiv:1806.10474 [cs, eess, stat]* .

Donahue, C., McAuley, J. and Puckette, M. (2019), ‘Adversarial Audio Synthesis’, *arXiv:1802.04208 [cs]* .

Dong, H.-W., Hsiao, W.-Y., Yang, L.-C. and Yang, Y.-H. (n.d.), 'MuseGAN: Multi-Track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment', p. 8.

Engel, J., Resnick, C., Roberts, A., Dieleman, S., Norouzi, M., Eck, D. and Simonyan, K. (2017), Neural audio synthesis of musical notes with WaveNet autoencoders, *in* D. Precup and Y. W. Teh, eds, 'Proceedings of the 34th International Conference on Machine Learning', Vol. 70 of *Proceedings of Machine Learning Research*, PMLR, pp. 1068–1077.
URL: *http://proceedings.mlr.press/v70/engel17a.html*

Gatys, L. A., Ecker, A. S. and Bethge, M. (2015), 'A Neural Algorithm of Artistic Style', *arXiv:1508.06576 [cs, q-bio]* .

Generating Long-Term Structure in Songs and Stories (n.d.), https://magenta.tensorflow.org/2016/07/15/lookback-rnn-attention-rnn.

He, K., Zhang, X., Ren, S. and Sun, J. (2015), 'Deep Residual Learning for Image Recognition', *arXiv:1512.03385 [cs]* .

Hinton, G. E. (2006), 'Reducing the Dimensionality of Data with Neural Networks', *Science* **313**(5786), 504–507.

Hochreiter, S. and Schmidhuber, J. (1997), 'Long Short-Term Memory', *Neural Computation* **9**(8), 1735–1780.

Huang, C.-Z. A., Vaswani, A., Uszkoreit, J., Shazeer, N., Simon, I., Hawthorne, C., Dai, A. M., Hoffman, M. D., Dinculescu, M. and Eck, D. (2018), 'Music Transformer', *arXiv:1809.04281 [cs, eess, stat]* .

Ioffe, S. and Szegedy, C. (2015), 'Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift', *arXiv:1502.03167 [cs]* .

Jeong, D., Kwon, T., Kim, Y. and Nam, J. (2019), Graph neural network for music score data and modeling expressive piano performance, *in* K. Chaudhuri and R. Salakhutdinov, eds, 'Proceedings of the 36th International Conference on Machine Learning', Vol. 97 of *Proceedings of Machine Learning Research*, PMLR, pp. 3060–3070.
URL: *http://proceedings.mlr.press/v97/jeong19a.html*

Ji, S., Luo, J. and Yang, X. (2020), 'A Comprehensive Survey on Deep Music Generation: Multi-level Representations, Algorithms, Evaluations, and Future Directions', *arXiv:2011.06801 [cs, eess]* .

Kingma, D. P. and Ba, J. (2017), 'Adam: A Method for Stochastic Optimization', *arXiv:1412.6980 [cs]* .

Kingma, D. P. and Welling, M. (2014), 'Auto-Encoding Variational Bayes', *arXiv:1312.6114 [cs, stat]* .

Lucas, J., Tucker, G., Grosse, R. and Norouzi, M. (2019), 'Understanding Posterior Collapse in Generative Latent Variable Models', p. 16.

Luong, M.-T., Pham, H. and Manning, C. D. (2015), 'Effective Approaches to Attention-based Neural Machine Translation', *arXiv:1508.04025 [cs]* .

Luque, S. (2009), 'The Stochastic Synthesis of Iannis Xenakis', *Leonardo Music Journal* **19**, 77–84.

Minsky, M. (1968), 'WHY PROGRAMMING IS A GOOD MEDIUM FOR EXPRESSING POORLY UNDERSTOOD AND SLOPPILY-FORMULATED IDEAS', https://web.media.mit.edu/˜minsky/papers/Why%20programming%20is–.html.

Nielsen, M. A. (2015), 'Neural Networks and Deep Learning'.

Oore, S., Simon, I., Dieleman, S., Eck, D. and Simonyan, K. (2018), 'This Time with Feeling: Learning Expressive Musical Performance', *arXiv:1808.03715 [cs, eess]* .

Performance RNN: Generating Music with Expressive Timing and Dynamics (n.d.), https://magenta.tensorflow.org/performance-rnn.

Roberts, A., Kayacik, C., Hawthorne, C., Eck, D., Engel, J., Dinculescu, M. and Nørly, S. (2019), Magenta studio: Augmenting creativity with deep learning in ableton live, *in* 'Proceedings of the International Workshop on Musical Metacreation (MUME)'.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A. and Chen, X. (2016), 'Improved Techniques for Training GANs', *arXiv:1606.03498 [cs]* .

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M. and Monfardini, G. (2009), 'The graph neural network model', *IEEE Transaction on Neural Networks* pp. 81–102.

Sengupta, S., Basak, S., Saikia, P., Paul, S., Tsalavoutis, V., Atiah, F., Ravi, V. and Peters, A. (2019), 'A Review of Deep Learning with Special Emphasis on Architectures, Applications and Recent Trends', *arXiv:1905.13294 [cs, stat]* .

Shaw, P., Uszkoreit, J. and Vaswani, A. (2018), 'Self-Attention with Relative Position Representations', *arXiv:1803.02155 [cs]* .

Sutskever, I., Vinyals, O. and Le, Q. V. (2014), 'Sequence to Sequence Learning with Neural Networks', *arXiv:1409.3215 [cs]* .

The MIDI Association - Standard MIDI Files (n.d.), https://www.midi.org/specifications/file-format-specifications/standard-midi-files.

Turing, A. (1950), 'Computing machinery and intelligence', p. 21.
URL: *http://www.loebner.net/Prizef/TuringArticle.html*

Understanding LSTM Networks – Colah's Blog (n.d.), https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. and Kavukcuoglu, K. (2016), 'WaveNet: A Generative Model for Raw Audio', *arXiv:1609.03499 [cs]* .

van den Oord, A., Kalchbrenner, N., Vinyals, O., Espeholt, L., Graves, A. and Kavukcuoglu, K. (2016), 'Conditional Image Generation with PixelCNN Decoders', *arXiv:1606.05328 [cs]* .

van den Oord, A., Vinyals, O. and Kavukcuoglu, K. (2018), 'Neural Discrete Representation Learning', *arXiv:1711.00937 [cs]* .

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. and Polosukhin, I. (2017), 'Attention Is All You Need', *arXiv:1706.03762 [cs]* .

Wang, C.-i. and Dubnov, S. (n.d.), 'Guided Music Synthesis with Variable Markov Oracle', p. 8.

Wyse, L. (n.d.), 'Mechanisms of Artistic Creativity in Deep Learning Neural Networks', p. 8.

Xie, S., Girshick, R., Dollár, P., Tu, Z. and He, K. (2017), 'Aggregated Residual Transformations for Deep Neural Networks', *arXiv:1611.05431 [cs]* .

Yang, L.-C., Chou, S.-Y. and Yang, Y.-H. (2017), 'MidiNet: A Convolutional Generative Adversarial Network for Symbolic-domain Music Generation', *arXiv:1703.10847 [cs]* .

Yang, L.-C. and Lerch, A. (n.d.), 'On the evaluation of generative models in music', p. 13.

Yu, L., Zhang, W., Wang, J. and Yu, Y. (n.d.), 'SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient', p. 7.