UNIVERSITY OF PRETORIA

MASTERS THESIS

Comparision of Adversarial and Non-adversarial Music Generative Models

By
Moseli Mots'oehli

Submitted in partial fulfillment of the requirements for the degree
Masters in Information Technology (Big Data Science)
in the Faculty of Engineering, Built Environment and Information Technology
University of Pretoria, Pretoria

June 21, 2019

Comparision Of Adversarial And Non-adversarial Music Generative Models

by

Moseli Mots'oehli
E-mail: u12061019@tuks.co.za

Abstract

Algorithmic music composition is a way of composing musical pieces with minimal to no human intervention. While recurrent neural networks are traditionally applied to many sequence-to-sequence prediction tasks, including successful implementations on music composition, their standard supervised learning method based on input to output mapping leads to a lack of creativity and variety. These models can therefore be seen as potentially unsuitable for tasks where a level or artistry is required, such as music. Generative adversarial networks, on the other hand, learn the generative distribution of data, and are used in tasks that require a level of creativity. This work implements and compares adversarial and non-adversarial training of recurrent neural network music composers on MIDI data. The resulting generated music samples are evaluated by volunteer human listeners, and their preferences are recorded to test whether adversarial training produces music samples that are more pleasing to listen to. The evaluation indicates that adversarial training does produce more aesthetically pleasing music.

Keywords: Music generation, MIDI, Generative adversarial networks, LSTM.

Supervisors : A.S.Bosman
Prof. J.P. de Villiers
Department : Department of Computer Science
Degree : Master of Information Technology (Big Data Science)

"Music expresses that which cannot be said and on which it is impossible to be silent."

-Victor Hugo (1864)

"Every artist was first an amateur"

-Ralph Waldo Emerson

Acknowledgments

I am grateful to the following people and institutions for the helping hand they provided during the two years I spent studying towards the completion of my master's degree:

- My supervisors Anna Bosman and Pieter de Villiers at the University of Pretoria, for allowing me to work on this topic under their supervision and the substantial amount of information and guidance they provided throughout the process of completing this dissertation;
- My family and friends, for the undying support and motivation they provided to help me go through the hard times I faced;
- Christopher Olah, Thalles Silva and Florian Colombo each for the generous permission to use neural network images from their work;
- The National Research Foundation of South Africa (NRF) and FirstRand for the partial financial support towards funding my studies.

Contents

List of Figures

List of Tables

Glossary

ANN: Artificial neural network
BAcc: Balanced accuracy
BPTT: Back propagation through time
CNN: Convolutional neural network
CV: Cross validation
EM: Earth mover
FFNN: Feed forward neural network
FN: False negative
FP: False positive
GAN: Generative adversarial network
GRU: Gated recurrent unit
KL: Kullback–Leibler
LSTM: Long-short term memory neural network
MIDI: Musical instrument digital interface
MLE: Maximum likelihood estimation
MOS: Mean opinion score
MSE: Mean square error
NLP: Natural language processing
NN: Neural network
RELU: Rectified linear unit
RL: Reinforcement learning
RNN: Recurrent neural network
TN: True negative
TP: True positive
WGAN: Wasserstein generative adversarial network

Chapter 1

Introduction

1.1 Motivation

Music composition, like most art forms, has for a long time been a skill specific to human beings. It has an intuitive side to it that is necessary to determine which pitches create harmorny together, what chords can be played after a certain note, or what note progressions are in violation of intrinsic musical theory. With the recent successes in neural network modeling of predictive natural behaviour and generative models, there have been good applications of modelling note progression probabilities for music generation.

The two dominant approaches to neural music generation are adversarial training [36, 37, 46, 56], and sequence-to-sequence recurrent networks [57, 58, 65], each with its merits. Although waveforms have been shown to be a viable way to generate audio not necessarily specific to music [19], it is symbolic representations that are favoured in literature for the task of music generation [37, 46, 53, 55, 57]. Owing to the existing lack of out-right comparisons between adversarial and non-adversarial training for music generation, the aim of this study is to compare music samples generated by two generative models, one trained in an adversarial setting, and the other in a non-adversarial setting, using musical instrument digital interface (MIDI) data.

1.2 Problem Statement

This work strives to demonstrate two points, namely: (1) That generative adversarial network (GAN)s with long-short term memory neural network (LSTM) cells can be used to generate polyphonic music that is realistic, creative and pleasing to listen to, and (2) that generative adversarial models with LSTM cells perform better than an identical non-adversarial LSTM based generator. An LSTM based neural network will be trained in an adversarial setting to generate music in MIDI format, and compared to an LSTM encoder-decoder network [14], that is not trained in an adversarial setting. Although adversarial training is much more complex in comparison to the encoder-decoder configuration for sequence-to-sequence models, GAN's ability to model note progression by sampling a latent space leads to a more diverse generator. A Wasserstein generative adversarial network (WGAN) [42] is implemented instead of the maximum likelihood estimation (MLE) based GAN to ensure stable adversarial training. Although MIDInet [46], a GAN based convolutional neural network (CNN) music generator, has been shown to produce better results compared to melodyRNN [58] models that deploy recurrent neural network (RNN) cells, the two networks implement two different generator types, CNN and RNN respectively, so the study provides no evidence to support the hypothesis that GAN training produces superior results in the music generation domain.

The musical data used in this work is in MIDI format. The simplicity inherent in pre-processing MIDI data as compared to pre-processing raw audio made MIDI a more suitable choice of music representation for the training data. To ensure music quality is not negatively affected by data representation, a common 2D state-matrix representation [40] of note progression is adopted for both training configurations. Although multi-track notes are captured, for the purpose of comparing the network's ability to model note progression, it is trivial to also learn multi-track probability distributions, and instead assume a MIDI type 0 file explained in Section 5.2 on decoding the resulting music and playback. The background of the architectures used in this study will also be explored, beginning with fully connected feed forward neural networks, activation functions, convolutions, recurrent networks, LSTM cells, and adversarial training. Standard music evaluation methods are used to come to a conclusion.

1.3 State of the Art

Despite much progress in music generation using deep learning, majority of methods focus only on note and chord progression transition probabilities when using a symbolic data representation such as MIDI . Majority of the work focusing on MIDI implement some form of LSTM [36], while the best networks on raw audio generation implement a CNN [19]. In Chapter 2, existing neural network (NN) implementations and their results for music generation are discussed.

1.4 Dissertation Layout

Chapter 2 explores existing recurrent and non-recurrent neural network implementations for music generation, and shows how the work in this thesis relates to existing literature.

In Chapter 3, the basic building blocks of deep neural networks are discussed, followed by a discussion on recurrent neural networks, which are used for modelling temporal data.

In Chapter 4, other non-recurrent neural network architectures such as convolutional neural networks used for music generation are discussed together with an introduction to adversarial training.

In Chapter 5, the MIDI data representation is discussed and motivation is given as to why it is preferred over other music representations for neural network training.

Chapter 6 contains the architectural details of the two generative models to be implemented and compared. The procedure for representing MIDI data is also discussed.

Chapters 7 and 8 together form the experimental setup and analysis of the results thereof, followed by the conclusion in Chapter 9.

Chapter 2

Neural Music Generation

The goal of algorithmic music generation is to be able to develop systems that enable automation of the composition process, while still achieving results comparable to human generated music. Although music as an art form has existed for millennia, the earliest publications on algorithmic composition are only as old as 1960 [52]. It was only towards the mid 1970s that significant interest and research was put into algorithmic music generation. There are different approaches to algorithmic music generation such as using mathematical models (stochastic processes) [54], grammar based methods [53], learning algorithms [55–58] and evolutionary methods [23]. However, the most promising results have come from the learning algorithms in recent years, in particular deep learning neural networks. The focus of this study is on music composition using artificial neural network (ANN) learning algorithms [19, 46].

A number of inventions in the deep learning domain contributed to majority of the work performed in neural music generation. The LSTM network [15] is well suited to successful learning of sequential data such as audio, and has the capability to recall notes generated a number of time steps back by solving the vanishing gradient dilemma that other RNNs suffered from. GANs [33] are especially useful for generating realistic data while reducing overfitting, and have been found to produce creative art [34–36]. Majority of the music generating neural networks to date are trained on either jazz or classical music, and only the piano track is used or all other tracks are played back on piano. SeqGAN [36] is a hybrid GAN between deep learning and reinforcement learning (RL), this model uses a RL generator agent to guide the generative

learning. By using RL with the discriminator providing a reward function, seqGAN [36] is able to out-perform standard MLE based GANs in music generation.

Colombo and Gerstern [55] proposed BachProp, an LSTM based network for learning note progression independent of note representation. They propose a three layered LSTM architecture to model notes, their timing, and duration by conditioning the two other attributes on the current note per time step. Although Colombo and Gerstern use a normalized MIDI representation of all training songs for training, they neglect to indicate how the network is representation invariant, as it assumes MIDI input data. Like Colombo and Gerstern, Olof [37] introduced continuous-recurrent GAN (CRNNGAN) for the same task, and adopted a similar network structure with three stacked LSTM layers in the generator network to enable learning of high complexity notes, chords and melody with an additional MIDI feature (note intensity) over BachProp. Unlike BachProp [55], CRNNGAN is trained in an adversarial setting. Due to their-three layered generator and continuous representation, CRNNGAN is limited to producing only up to three different tones per time step, hence produces music that is not rich in polyphony.

Bengio et al. [57] show that a gated recurrent unit neural network can be used to achieve results at least comparable to those of more sophisticated gated networks such as LSTM, keeping all training parameters equal. They train both models on piano-track MIDI data, and evaluate generated samples for polyphony. However, comparison over multiple datasets produced inconclusive results. MelodyRNN [58] is a collection of RNN models (LookbackRNN and AttentionRNN) for polyphonic music generation trained on MIDI data. The LookBackRNN implements a look-back mechanism to help the network recall very long dependencies in generation, and the attentionRNN implements an attention mechanism [25] for increased note repetition to improve rhythm.

WaveNet [19] introduced by the Google DeepMind team is a completely probabilistic network that uses the same architecture as PixelCNN [28] to generate raw audio waveforms from a dataset of mp3 files with multiple tagged genres. WaveNet was developed mainly for the task of text to speech synthesis, and uses multiple layers of time dilated convolutional networks instead of more traditional and suiting sequence

models such as RNNs. The DeepMind team do this to avoid the long training time required for RNNs. However, WaveNet's generator is not trained in an adversarial setting, and no quantitative results on music generation are reported by the authors.

MidiNet [46] expands on this work using MIDI files instead, and train CNNs in an adversarial setting. Both networks learn note progression by sequentially conditioning future notes on the distribution of previously generated notes by using dilated convolutions. Although MidiNet outperforms standard RNNs, and produces music that is considered more varied and pleasing to listen to than both the LookbackRNN and AttentionRNN, it is unclear as to how it would compare to LSTM and gated recurrent unit (GRU) based networks which are superior to standard RNNs on very long sequence tasks. This work implements an encoder-decoder LSTM network in the same fashion as BachProp[55], and an LSTM based GAN as in [36, 37, 56].

The majority of the work mentioned above use MIDI training data, and only a few train from raw waveforms. The most common way of representing the MIDI features to be learned is using a 2D matrix [56] of binary entries where pitch is on one axis, and the other axis represents time in ticks. In some cases the real continuous values from the MIDI messages are entries in the 2D matrix [37, 65]. Some work has been done in learning multi-track note progressions, although results show lack of synchrony and cross track dependency learning. Chu et al. [70] implement a sequence-to-sequence model of four stacked LSTM layers to model multi-track note progression with each LSTM cell generating its own track's output. Other notable neural network approaches for music generation include variational autoencoders [60], restricted Boltzmann machines [61], and deep belief neural networks [47] to learn note progression probabilities that boost rhythmic scores. They show how the resulting model can be used as a prior distribution for training an RNN for polyphonic music.

In the next chapter, background on the neural network architectures used for music generation and their components is discussed.

Chapter 3

Feed Forward and Recurrent Neural Networks

In this chapter, the basic building blocks of NNs are discussed together with common architectures relevant to temporal data. Section 3.1 covers the building blocks of multi-layer neural networks and how the components are connected together to allow learning. RNNs, and how their architechture allows for modelling of temporal dependencies, are then discussed in Section 3.2. The chapter concludes by providing an overview of the encoder-decoder configuration of training sequence-to-sequence NNs in Section 3.3.

3.1 Multi-layer Feed-Forward Neural Networks

ANNs are computational units inspired by the biological neural networks that make up the human brain. The basic structure of ANNs consists of an input layer of multiple neurons interconnected to one or more hidden layers through trainable weights, bias term and non-linearity activation functions, and finally connecting to an output layer. These networks are trained to adjust the collective weights of all layers so as to produce output vector $\bar{y} = (y_1, y_2, ..., y_m), m \in \mathbb{Z}$ that optimizes a predefined objective function given input vector $\bar{x} = (x_1, x_2, ..., x_n), n \in \mathbb{Z}$. The forward pass of information in a supervised ANN trained using gradient descent [69] is followed by back propagation [3] of the output errors back to all layers, and adjustment of the network weights. ANNs with only a few hidden layers and limited neurons within each

layer such as the four layered neural network in Figure 3.1 are capable of learning a limited degree of complexity. To expand on this, more hidden layers can be added to make the network deeper and denser with interconnections, hence the name multi-layer or deep neural network (DNN).

Within each neuron, the activation function used determines the type of relationships the network can model. A DNN with only linear activation functions is limited to learning only linear combinatory mappings of the inputs to the outputs. To remedy this problem, nonlinear activation functions such as sigmoid $\sigma(x)$, rectified linear unit (RELU) [7], and $\tanh(x)$ are introduced throughout the network. Given a deep enough network of bounded width and suitably nonlinear activation functions, the multi-layer neural network is said to be a universal continuous function approximator [8]. The most basic configurations of these networks are however not well-equipped to handle temporal or spatial dependencies in the input data, which has led to the invention of more domain specific architectures for time series prediction and image recognition. All neural network variants below are inspired and borrow much from the original multi-layer feed-forward neural network.

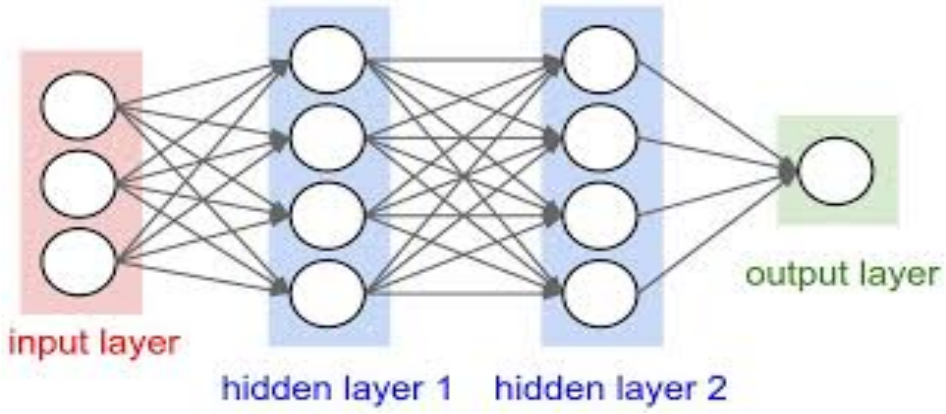


Figure 3.1: *A fully connected feed forward ANN that takes 3 inputs, with 2 hidden layers and an output layer of 2 neurons. Source: [2]*

3.1.1 Important Background Concepts

Throughout the chapters in this work, a number of key concepts will be mentioned without explanation. In this section, some of these concepts are breifly discussed starting with training, test and validation dataset splits in Subsection 3.1.1.1, regularization techniques in Subsection 3.1.1.2, and finally evaluation matrices and loss functions in Subsection 3.1.1.3.

3.1.1.1 Training, Validation, and Test Datasets

In supervised machine learning, it is common to split all available data into three subsets, namely: training, validation, and test datasets. The training and test dataset are completely disjoint sets, while the validation dataset is only independent of the training set for each complete pass of the entire training set. The validation set is kept away from the model during training, and is used to assess how well the model performs on a reasonably similar data distribution

The reason for spliting the dataset is to evaluate whether the model generalizes well and does not overfit. The training set offers supervised examples to the model , used for network weight tuning. The validation set is used to measure unbiased performance on the training dataset for network parameter tuning during cross validation. The test set is used to measure how well the network generalizes on an unseen dataset generated by a reasonably independent yet similar distribution compared to that of the training dataset. An overly complex model normally performs better on the training set than it does on the test set, this phenomenon is termed overfitting and is remedied by using regularization, discussed next.

3.1.1.2 Regularization

Regularization methods are methods used to discourage ANNs from using more training weights than is necessary, since excessive weights may lead to overfitting. In other words, regularization either penalizes an overly complex model or forces a NN to use only a random portion of its weights per training step. Examples of the former

method are adding the first (L1) or second (L2) norms [9] of the network weights to the loss function. The latter method is known as dropout [10]. Dropout reduces model complexity by randomly setting a portion of the total number of network weights to zero with probability α for each training epoch. A dropout probability of $\alpha = 0.2$ is used in the experiments in this dissertation.

3.1.1.3 *Evaluation Metrics and Loss Functions*

During and after training of NNs, it is important to evaluate how well the learned network weights fit the data. Common metrics for evaluating performance include: accuracy (propotion of correct prediction instances out of all prediction instances), recall score (proportion of positive instances correctly classified out of all positive predictions), and balanced accuracy (BAcc), given by the equation:

$$BAcc(y, \hat{y}) = \frac{1}{2} \times \left[\frac{TP}{TP + FN} + \frac{TN}{TN + FP} \right], \tag{3.1}$$

where true positive (TP), true negative (TN), false positive (FP), and false negative (FN) are given by the total prediction instances in the labelled data satisfying the confusion matrix in table 3.1:

	Ground Truth Labels	
Predicted Labels	$y = 1$	$y = 0$
$\hat{y} = 1$	TP	FP
$\hat{y} = 0$	FN	TN

Table 3.1: *Classification confusion matrix*

Loss functions, also known as objective functions, are used to measure how large the scalar error is between the model's current predictions and the expected output. During training, the goal is to minimize this loss to an acceptable level. Common loss functions include: mean square error (MSE) (mean squared difference between the predictions and the real regression target values). The cross entropy loss 3.2, used for classification tasks, minimizes the difference between the actual and the target output distributions and is defined by the equation:

$$H(y, \hat{y}) = -\sum_i y_i \log(\hat{y}_i), \tag{3.2}$$

where y_i and $\hat{y}_i$ represent the true label and the predicted activation probability respectfully. This is an expectation over the true distribution of y_i.The values outputed by the loss function are used by the model in fiting the data in a process called back propagation [3], discussed in the next section.

3.1.2 Back Propagation

In the forward pass of information in a ANN, an input vector x_{i-1} is multiplied by a weight matrix W_{i-1}, then a bias term b_{i-1} is added, and a non-linearity activation function f is applied to the result to get the output of the next layer in the network as shown in the equation below:

$$y_i = f(x_{i-1}W_{i-1} + b_{i-1}), \tag{3.3}$$

for layer $i = 1, 2, ..n$. This process continues until the final output layer where the model error on estimation is calculated using a loss functions such as the MSE, or cross-entropy loss.

The error term is then propagated back to all layers to adjust the model weights in such a manner that corrects for the error by minimizing the loss function. Gradient descent is applied to find the minimum of the objective function. Since the objective function is a function of multiple activation functions, the chain rule of differentiation is used in back propagation [3] to calculate the gradients of the network with respect to all the network weights. The process of calculating gradients and performing weight updates is repeated over the entire training dataset multiple times until an acceptable number of iterations or acceptable loss value is reached.

3.1.3 Activation Functions

Activation functions are functions $f(xW + b)$ within the neurons of a NN that take as input the weighted outputs of a previous layer and produce an output. Different activation functions are used in different layers of the NN depending on the properties

of the target being approximated, and the properties of the activation itself. Common activation functions used in deep learning are discussed below. The linear activation function is giveb by: $y = f(xW + b) = xW + b$. Notice that $\frac{\partial f}{\partial x} = W$. This implies that back propagation will result in constant gradient to the weights regardless of the input x.

3.1.3.1 Sigmoid

The sigmoid activation function is defined by:

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{3.4}$$

When x is large and tends to ∞ , e^{-x} approaches 0, and so $\sigma(x)$ tends to 1. When x tends to $-\infty$, e^{-x} tends to ∞ and so $\sigma(x)$ approaches 0. This is shown by the sigmoid function in Figure 3.2. Unlike the linear activation function, $\sigma(x)$ is bounded between 0 and 1. The gradient is given by:

$$\frac{\partial \sigma}{\partial x} = -(-e^{-x})(1 + e^{-x})^{-2} = \frac{\sigma(x)}{(1 + e^{x})}, \tag{3.5}$$

which is a function of x unlike in the linear activation. When $x \to \infty$, $\sigma(x) \to 1$ and so $\frac{\partial \sigma}{\partial x} \to 0$. Similarly, when $x \to -\infty$, $\frac{\partial \sigma}{\partial x} \to 0$. This property of the sigmoid activation function is a major problem in s since it implies that for relatively large inputs to a layer, the gradients tend to zero, this multiplied with the error terms results in insignificantly small to no weight updates, and the network fails to learn. The phenomenon is termed the vanishing gradient problem [5]. The vanishing gradient problem is common for bounded functions such as sigmoid and hyperbolic tangent. The vanishing gradient is one of the main problems in modelling temporal dependencies in sequential data such as music.

3.1.3.2 Hyperbolic Tangent

The hyperbolic tangent function (tanh) squashes its inputs to a range between -1 and 1, and like the sigmoid activation saturates towards the two extremes with gradients tending to zero for large negative and positive x as shown in Figure 3.2. However,

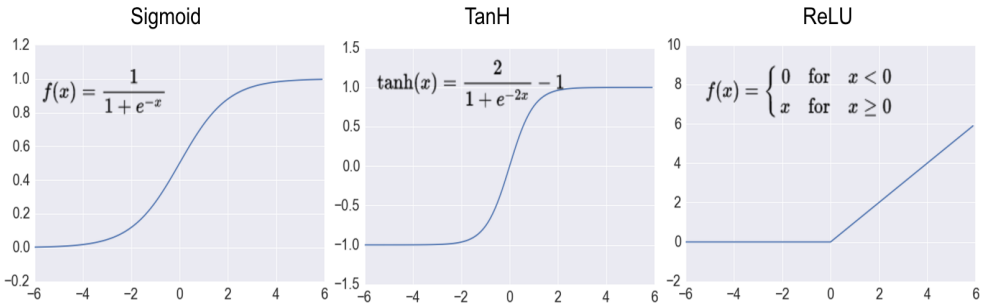


Figure 3.2: *Common activation functions. Source: [4]*

unlike sigmoid, tanh output is centred around zero. The tanh activation can be expressed as a scaled sigmoid function:

$$\begin{aligned} \tanh(x) &= \frac{e^x - e^{-x}}{e^x + e^{-x}} \\ &= \frac{1 - e^{-2x}}{1 + e^{-2x}} \\ &= (1 - e^{-2x})\sigma(2x) \\ &= 2\sigma(2x) - 1, \end{aligned} \tag{3.6}$$

where $2\sigma(2x)$ is in range (0,2) , and the -1 centers the outputs between -1 and 1. It is for this reason that tanh is favoured in the hidden layers over the sigmoid activation in sequence models, as it is more resilient to vanishing gradients, since it saturates slower than sigmoid and has higher gradients towards extreme values [11].

3.1.3.3 Rectified Linear Unit

RELU as depicted in Figure 3.2 is a modified linear activation, and only positive inputs cause the neurons to fire. All negative inputs are mapped to zero and the positive inputs to themselves so that RELU has range [0,∞). RELU is given by $f(x) = \max(0, x)$. Because RELU activations do not explode exponentially, and it is very easy to compute in both the forward and backward propagation, it has become the most widely used activation function for deep learning models in domains such as image classification. In cases where the target variable assumes both negative and positive

values, RELU alone is unable to model negative values as an output activation, so it is used in partnership with tanh or a linear activation. Most deep learning models employ RELU between the hidden layers.

LeakyReLu's formulation is given by $f(x) = \alpha \times x$, for $x < 0$, and $f(x) = x$, for $x >= 0$, where $0 < \alpha < 1$. LeakyReLu a modified version of RELU that allows a very small proportions of negative inputs to cause the neuron to fire and avoids the dead RELU problem that occurs when positive weights and negative inputs always result in a zero RELU activation and halt learning.

3.2 Recurrent Neural Networks

A RNN [58] is a specialized type of ANN that, unlike feed forward neural network (FFNN)s, has recursive connections between its layers as shown in Figure 3.3. RNNs are designed to handle sequential inputs in such a way that temporal dependencies are preserved in memory and have an influence on later outputs in a time series regression or classification task. RNNs have a recurrent connections in the hidden layers $h_t = f(x_t, h_{t-1})$ that theoretically is able to recall all information from previous time-steps. Owing to this attribute, RNNs are well suited for sequential data such as text, audio and video comprehension where note and word probabilities $p(y_t|x_t, y_{t-i})$ for $i = 1, 2, .., n$ are temporally conditioned on previous notes or words.

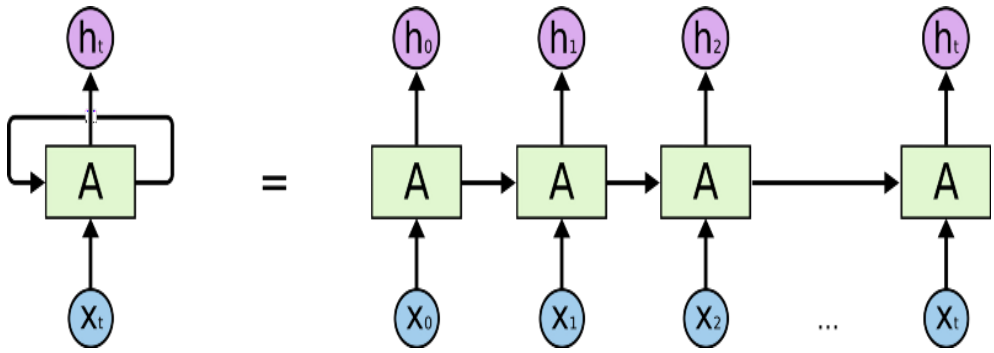


Figure 3.3: *A recurrent neural network cell showing information flow over time. Source:[17]*

The RNN architecture is in practice unable to learn long term temporal dependencies due to vanishing error gradients in back propagation. In RNNs, error terms are not

just propagated from output to input layer, but also through all time-steps by using the back propagation through time (BPTT) [6] algorithm discussed in Section 3.2.1 to follow.

Despite all its shortcomings, the base RNN still performes relatively well on temporally short sequence based tasks [58]. There are modifications to the standard RNN cell and its training that overcome the RNN problems ranging from changing initialization methods, introducing non exponential activation functions such as RELU for vanishing gradients, to entire reconstructions of the recurrent cell unit.

3.2.1 *Back Propagation Through Time*

BPTT [6] is a gradient based algorithm for training recursive models such as RNN. The underlying concept in BPTT is similar to that of normal back propagation for FFNNs. In BPTT, the RNN is unrolled in time, and each time-step has an input, hidden state and an output, all presented to the network in the order they appear in the time series. The network shares weights across all time-step copies of the recurrent cell. In the forward pass of data, the network is unrolled and time-step specific output errors are calculated, the network is then rolled back up and the error terms are accumulated to derive a common network error. As in standard back propagation, the cost function must be expressible in terms of all the network weights, biases and activation functions, and must be differentiable. Once the gradients are calculated, all the weights are updated to minimize the cost. This process continues for n passes of the training dataset (epoch), or until an acceptable loss is achieved.

BPTT is used extensively in most sequence based NNs, since it provides an efficient way to calculate all the gradients in a complex objective. However, in cases where the input or output sequences presented to the network are very long, gradients tend to saturate and hinder learning. Truncated BPTT [13] is a variant of BPTT where errors are not accumulated for the entire sequence in order for an update to occur. Instead, a window is set where an update to weights in time-step t_i depends only on the gradients up to time-step t_{i+k}, where k is a reasonably small window. This is done to remedy vanishing and exploding gradients, while at the same time still allowing

the network to not overlook any long term temporal dependencies in such a way that learning gaps are created. The process of determining such a suitable window k can be as difficult as determining the right hyper-parameters for a NN, and is problem specific. The next section discusses a commonly used recurrent network called LSTM.

3.2.2 Long-Short Term Memory Neural Networks

LSTM neural networks were developed by Hochreiter and Schmidhuber [15]. An LSTM cell is an architectural improvement over the standard RNN unit in that it has gates that "decide" what new information from the current input step to include into the hidden cell state, what information from the hidden state to forget, a layer that aggregates all this information and acts on the other two layer's decisions, and an output layer similar to that of an RNN. These enhancements over the RNN cell enable the LSTM to better handle the vanishing and exploding gradient problem and hence makes it better suited for music generation, since it is able to keep up long term note and chord progression conditional probabilities during the training phase. In the forward propagation, the following equations are iterated over per input time-step $t = 1, ..., T$ per training epoch:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f), \tag{3.7}$$

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i), \tag{3.8}$$

$$\bar{c}_t = \tanh(W_{\bar{c}}[h_{t-1}, x_t] + b_{\bar{c}}), \tag{3.9}$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o), \tag{3.10}$$

$$c_t = f_t c_{t-1} + i_t \bar{c}_t, \tag{3.11}$$

$$h_t = o_t \tanh(c_t), \tag{3.12}$$

where W_i and b_i represent learnable network and bias weights respectively, h_t and x_t represent the recurrent network's hidden layer state and the current input respectively. The σ and tanh are activation functions that were discussed in Sections 3.1.3.1

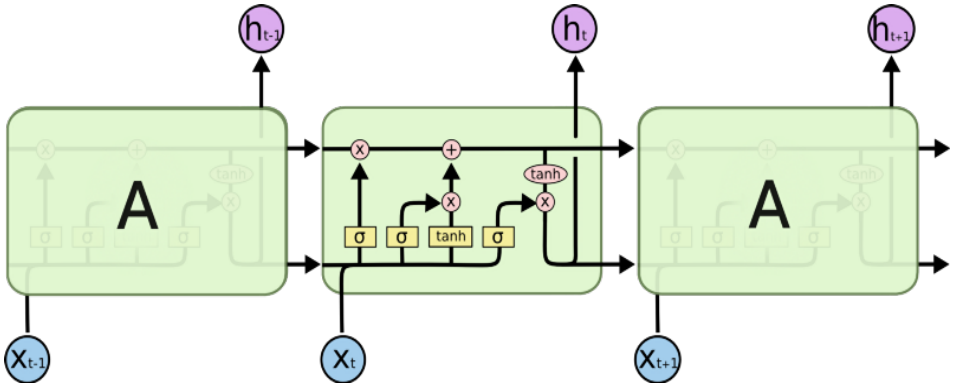


Figure 3.4: *An LSTM cell unrolled in time. Source: [17]*

and 3.1.3.2 respectively.

Equation 3.7 decides how much of the information from previous states to forget, Equations 3.8 and 3.9 together make up the gate that decides how much of the new information from input x_t to add to the cell state. Equation 3.10 decides which parts of the cell state will be output by the network at time t. Equation 3.11 does the actual update of the new cell state with a weighted sum of information to incorporate from the previous cell state, and information to update from the proposed cell state. Finally Equation 3.12, calculates the output at time t is the hidden state, which is a filtered version of the cell state, filtered for what the network decided to output in Equation 3.10. Figure 3.4 shows the flow of information within an LSTM cell.

Although LSTM networks are known in theory to be able to prevent the vanishing gradients, they still suffer to some extent from the same problem in practice. Efforts have been made to remedy this through techniques such as stacking LSTM layers, capturing both the forward and backward flow of input information (Bidirectional recurrent neural networks [30]), creating peephole gates for more information retention, introducing skip connections over some LSTM layers [29], using specialized non-saturating activations such as leakyRelu, discussed in Section 3.1.3.3, gradient clipping to handle exploding gradients [51], adding an attention mechanism [25] and pointer generator neural networks [26] in the encoder-decoder LSTM configuration [14] discussed in Section 3.3, amongst others.

3.3 Encoder-Decoder Models

Encoder-decoder RNNs [14] were designed to specifically address learning problems where the input and output are sequences of varying lengths. The standard RNN is unable to do this while conditioning the output sequence on the entire input sequence. The encoder-decoder models consists of two RNNs or LSTMs that are trained together using a single objective function. The encoder learns to map variable length inputs to a fixed length vector which is used by the decoder as initial input, and is mapped to a variable length output. The encoder-decoder configuration of training sequence-to-sequence models is well suited for tasks such as music composition [55], text summarization [24], language text translation [20] and question answering where the input and output are sequences that each have temporal dependencies.

3.4 Conclusion

This chapter has covered FFNNs, RNNs and the encoder-decoder training configuration from a music generation perspective. Concepts such as activation functions, evaluation metrics, loss functions, regularization, and their properties were discussed to support the choice of model architectural components used in Chapter 6 of this work. The following chapter looks at how convolutional and adversarial neural networks work and how they have been successfully used in the domain of music composition.

Chapter 4

Convolutional and Adversarial Neural Networks

In this chapter, non-recurrent neural networks and adversarial training are discussed with respect to how they adapt to temporal data. Section 4.1 covers CNNs, and Section 4.2 discusses GANs.

4.1 Convolutional Neural Networks

Convolution is a mathematical operation that measures the amount of overlap between two functions f and g. CNNs derive their name from this operation and are responsible for much of the recent successes in image recognition and video tagging [66]. In a CNN, a kernel/filter f is slid over an image g, and a dot product is applied between elements of the image vector and those of the filter to learn convoluted lower dimensional features that preserve the spatial relations between the pixels in the original image. In this manner, the convolution operation is used to reduce the dimensionality of the image by learning kernels with good overlap on the image. CNNs may have one or more layers of convolution, pooling layers for dimensionality reduction and non-linearity activation functions with trainable weights and biases. A standard CNN is normally followed by a fully connected FFNN layers, and is trained through back propagation.

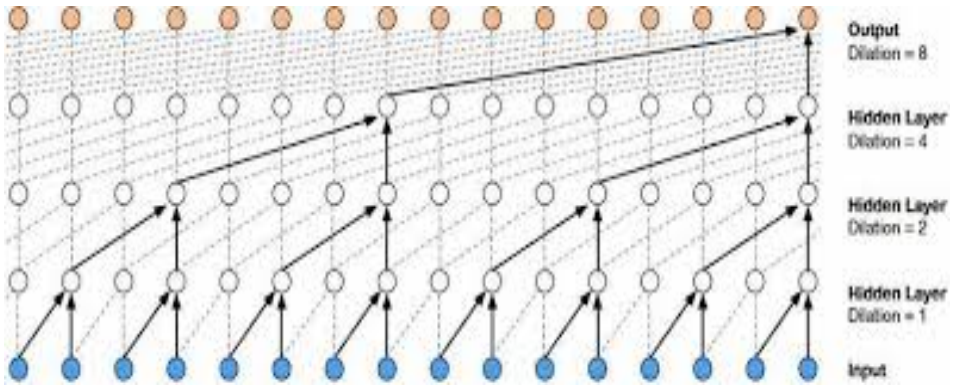


Figure 4.1: *A time dilated convolutional neural network. Source:[19]*

For music generation, CNNs are not as favoured nor suited as RNN and LSTM, as they assume spatial rather than temporal dependencies in the input data. However, CNNs are much easier to train since they have fewer parameters than RNN, FFNN and LSTM networks with the same number of hidden units. This is because CNNs have pooling layers that perform dimensionality reduction on the output of previous layers and because a single small kernel is repetitively applied to the entire input. To take advantage of the inexpensive training of CNNs for music generation, methods such as conditioning each note on the previous output note [19, 46], treating a time-frame of notes as an image and many more have been used [21]. For this study, only those CNN architectures that archived results comparable to recurrent models are discussed. One such model is Deepmind's Wavenet [19] that uses time dilated convolutions shown in Figure 4.1, to model conditional distributions of output frequencies, in essence mimicking how RNNs work to some extent.

4.2 Generative Adversarial Neural networks

First introduced by Ian Goodfellow et al. [33], GANs are generative models trained in an adversarial setting that aim to generate samples from an unknown distribution. The GAN system trains two networks (the generator and discriminator) in a zero-sum game fashion until a Nash equilibrium [31] is reached, where the generator generates samples so realistic and similar to the unknown training distribution samples that the

discriminator cannot tell them apart. The generator NN, $\bar{x} = G(z)$ accepts a noise signal z as input, and produces an output vector with the same dimensionality as the data whose generative distribution is modelled. Figure 4.2 shows a CNN based GAN used for image generation. The discriminator, $D(x)$ takes real data x and the generator's samples $\bar{x}$ as input during training and models the probability distribution of a sample belonging to the unknown real data generating distribution. The two networks are connected so that errors can be propagated back to the generator from the discriminator. The discriminator is in essence trained like a classifier that can tell the real and generated/"fake" data samples apart by producing a probability of an input sample being drawn from the real data. The two standard GAN networks are jointly trained over a *minmax* objective function given by:

$$min_G max_D V(D,G) = E_{x\sim P_X}[\log(D(x))] + E_{z\sim P_Z}[\log(1 - D(G(z)))]. \tag{4.1}$$

$G(z)$ and $D(x)$ can be any of the NN architectures mentioned in Sections 4.1, 3.2.2, 3.2, depending on the domain of application. For music generation, RNN and LSTM are natural choices since they were designed to handle time series data such as music notes. GANs are especially well suited for the task of music generation since they are generative models and their ability to model note progression probabilities using only random noise in the generator enable sthem to exhibit more creativity than FFNNs in their composition of audio sequences. However, GANs have been found to be unstable in standard training, and convergence to equilibrium is not always guaranteed [43]. Since their introduction, numerous variants [35, 41, 42] of GANs have been proposed that provide improvements in training simplicity and performance over the original implementation. Ian Goodfellow et al. [44] later released a paper on improved techniques for training GANs. This work focuses on Wasserstein GAN (WGAN) due to its qualities and ease of training. WGAN is discussed in the next section.

4.2.1 Wasserstein GAN

Wasserstein distance or earth mover (EM) distance is a measure of the amount of work required to move one probability distribution to another. While traditional

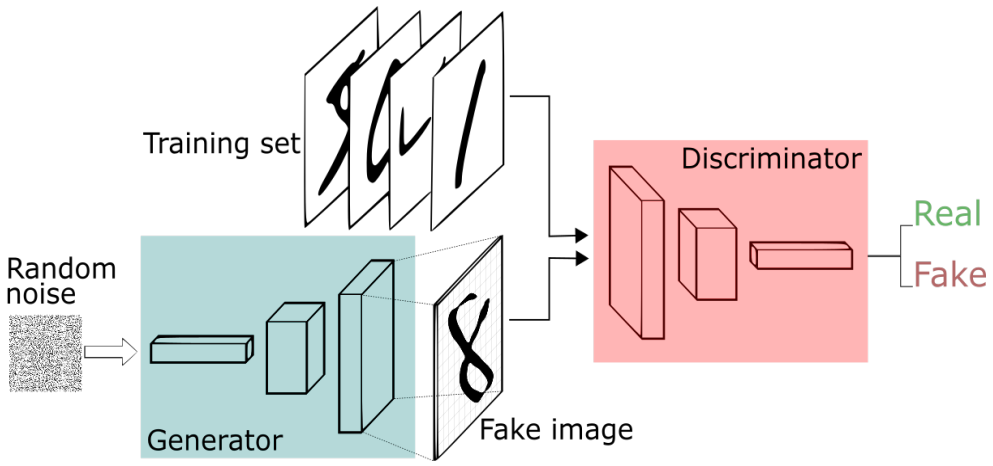


Figure 4.2: *A generative adversarial neural network with convolutional neural networks in both the generator and discriminator for handwritten digit image generation. Source: [18]*

GAN seeks a density distribution P_θ that maximizes the likelihood of samples from the distribution P_r to be modelled, WGAN minimizes the Kullback Leibler (KL) divergence distance which is a reasonable approximation to EM distance [42]. EM distance can be approximated by the equation:

$$W(P_r, P_\theta) = \inf_{\gamma in \Pi(P_r, P_\theta)} E_{(x,y)\sim\gamma}[||x - y||], \tag{4.2}$$

which has properties that ensure convergence in situations where other distance measures fail to converge. An example is the case in traditional GAN where the support of P_θ is drawn from a low dimensional latent space, which may intersect with that of the real data generating distribution P_r to a significant enough degree, so that most distance measures are invalid or are infinite. As a result of this, WGAN is much more stable than GAN, and needs less architectural hyper-parameter tuning of the generator and discriminator.

It has also been shown that unlike GAN that suffers from a dominant discriminator, the EM estimation in WGAN benefits from constant discriminator improvement, achieved by removing the sigmoid output activation that produces a probability of a generated sample being real in standard GAN, and using a linear activation instead, so that the distance between real and fake samples is as large and clearly defined as

possible from the beginning of training. In WGAN, the targets are also labelled differently from GAN, -1 and 1 instead of 0 and 1 for fake and real data, respectively. This has a benefit of leading to easy calculation of the loss. WGAN optimizes the following objective function:

$$min_G max_D V(D,G) = E_{x\sim P_r}[D(x)] - E_{z\sim P_\theta}[D(G(z))], \tag{4.3}$$

where P_r and P_θ represent the real and learned approximation of the data generating distribution respectively. $G(z)$ is the generator neural network, $D(x)$ the discriminator, X and Z represent the real data sample and the latent sampling vector respectively.

4.3 Conclusion

In this chapter, CNNs were breifly discussed followed by how their architecture have been modified to be able to model temporal data through dilated convolutions. GANs, which this work heavily depends on for the task of music generation, were also discussed. In the next section, music data representations commonly used for training NN are discussed. The chapter places more emphasis on the MIDI data representation which is the representation of choice for this dissertation.

Chapter 5

Data

Over the years, multiple musical representations for NN training have been proposed. In this section some of the most commonly used representations are discussed and motivation is given for the choice of data representation used for this work. In Section 5.1, the ABC textual music representation, is briefly explained. This is followed by a detailed discussion of the MIDI representation in Sections 5.2.

5.1 ABC

ABC notation is a simplified textual representation of music that uses letters: A to G to represent notes. It also uses other ASCII special characters and numbers to represent music attributes such as note length, key change events and more complex note behaviour. Because ABC is textual, when used as training data in neural music composition it is processed in a similar manner as natural language data. This means that a word or character level embedding is created by training a shallow NN to predict the characters or words surrounding a given word or character. The Word2Vec algorithms [45] are commonly used for creating word and character embeddings.

Once the embeddings are created, they are then fed to a music generation learning algorithm. However, training NN using ABC notation is more complex than using MIDI because, apart from the fact that MIDI data is already in an easy to parse "key: value" standardized format and ABC is not, ABC includes special characters such as

":,], #, |, [, %, _" that do not form a part of the natural language and so pose a problem during vector encodng using standard NLP tools. Also, because ABC to vector representations are commonly performed on a character level, the embedding size can be very large depending on the training corpus, making the entire training process memory inefficient. Figure 5.1 is an extract of "Speed of the Plough's" [71] ABC file showing attributes: title, meter, key, default unit note length, each on a separate line respectively. Note transcription begins with special character "|:".

```
T:Speed of the Plough
M:4/4
C:Trad.
K:G
|:GABc dedB|dedB dedB|c2ec B2dB|c2A2 A2BA| GABc dedB|dedB dedB|c2ec
B2dB|A2F2 G4:| |:g2gf gdBd|g2f2 e2d2|c2ec B2dB|c2A2 A2df:|
```

Figure 5.1: *ABC transcription of Speed of the Plough. Source: [71]. "|:" Is a repeat instruction until a stop instruction ":|" is encountered.*

5.2 MIDI

Musical instrument digital interface (MIDI) is a set of standards that outline how to connect digital music instruments, so they can communicate using a messaging layout that is standardized. These messages are termed MIDI messages, and encode instructions that can be decoded to produce sound by any digital instrument that conforms to the MIDI standard. The standard was drafted with simplicity and portability in mind, since a MIDI file does not contain the actual audio itself, but rather instructions to synthesize the original audio. In terms of music, each song is encoded in its own MIDI file, with each instruction to the synthesizer contained in a MIDI message. There are several MIDI message types: "note on" messages, "note off" messages, meta messages, control change messages, program change messages, and tempo change messages.

To assemble a fully functional MIDI file, all these messages are required, but for the purpose of modelling note progression and suitability to evaluate the model's ability

Channel	Note	Time	Type	Velocity
0	39	0	note_on	80
0	58	0	note_on	80
0	46	0	note_on	80
0	61	0	note_on	80
0	70	0	note_on	80
0	70	240	note_off	64
9	39	0	note_on	80
9	39	480	note_off	64
1	46	0	note_on	80
9	58	0	note_on	80

Table 5.1: *A sample MIDI file with note action messages presented in tabular form. The "Note" column represents pitch. Each of the file attributes listed in the table are discussed below.*

to generate audio, "note on" and "note off" messages on their own are sufficient, with other messages being set to default values. Each note event message contains several attributes that accompany it. Table 5.1 shows how note action messages are stored in a MIDI file.

As stated in Section 5.1, ABC vector representations are very large in size depending on the training corpus, making the training processes memory inefficient. This is not the case with MIDI since each pitch value can be represented by a binary encoded vector of length 128. Also, using ABC representation of music requires pre-training of the character embeddings, while this is not the case with the vector representation of MIDI data. These reasons have led to the use of MIDI and not ABC music data representation in this work. Subsections 5.2.1 to 5.2.4 below explain the four key attributes used in encoding each MIDI note message.

5.2.1 Channels

There are 16 available independent channels numbered from 0 to 15, through which each MIDI message can be transmitted to a MIDI enabled device. Each channel transmits messages from one or more tracks to a single musical instrument. Because the

channels are independent, multiple instruments can be played at the same time at different levels of loudness and varying tempos per channel. "Programme change" messages are used to switch between channels while transmitting messages.

5.2.2 Pitch

The pitch of a note, arguably the most important attribute in a musical piece, refers to the discretized frequency of sound produced by an instrument. The pitch attribute in a MIDI file assumes discrete values in the range [0,127]. Musical pitches are ordered and are identifiable in groups as low, middle or high pitches to the human ear. The 128 pitches are organized into 16 octaves with each pitch in an octave assigned an alphabet A to G in terms of ABC music notation discussed in Section 5.1. For the purpose of this study, only pitch progression and note message type are estimated, and the rest of the attributes are fixed. This is because note progression is a key determinant in the perceived quality of music generated.

5.2.3 Velocity

The velocity of a pitch refers to the force at which it is played. In terms of a physical piano, for example, differences in velocity on the same pitch would mean pressing the same key harder on one instance and softer on the other, hence creating two different levels of loudness. Velocity in MIDI takes on integer values from 0 to 127, where 0 is inaudible/silence and therefore can be used to represent a note-off event.

5.2.4 Time

The duration which each note holds is implicitly encoded in the MIDI message as delta time. This refers to the uninterrupted time between messages on a note of the same pitch. As a result, the duration of each note is determined by the delta time (in number of ticks) in its note-off message. The timing of each note is relative to all note-on and note-off messages of the same pitch and channel that precedes it. Although two MIDI files can have the exact same "note on" and "note off" messages including delta times, they can sound very different during playback due to the tempo messages

in the header and tempo change messages. As such, it is important to note that delta time depends on tempo. The tempo sets the pace of the song. The algorithm explained in Section 6.1 is used to normalize delta time for all MIDI files with different tempos.

5.3 Dataset

Since MIDI files are compact relative to the raw audio represented, there are numerous training dataset sources available online to choose from for the task of neural music generation. The Lahk MIDI dataset [72] is one of the most commonly used for this purpose. The dataset is a collection of 176,581 unique MIDI files of different genres and composers. Of the seven versions of the dataset, the "clean MIDI subset" version containing 17,257 song with filenames indicating song titles and artist was used. This dataset is of size 224MB compressed and 770MB uncompressed. Due to memory constraints, only 289 distinct polyphonic songs from 10 composers were used for training comprising only 10MB of disk space. Table 5.2 gives a statistical summary of the training data used for the different composers.

Composer	**Number of Songs**	**Number of Notes**
Beethoven	29	158 978
Billy Joel	117	947 296
Borodin	7	24 260
Diana Ross	3	39 022
Elton John	31	330 044
Elvis	5	29 776
Frank Sinatra	45	268 180
Liszt	16	53 366
Mendelssohn	15	79 516
Mozart	21	81 470
Total	**289**	2 011 908

Table 5.2: *Training dataset description by composer. Number of notes represent both note on and off messages*

5.4 Conclusion

In this section, common music data representations used in note progression modelling for music generation were discussed, and emphasis was placed on the MIDI representation that is used in this work. In the next chapter, the methodology followed for the study is discussed.

Chapter 6

Methodology

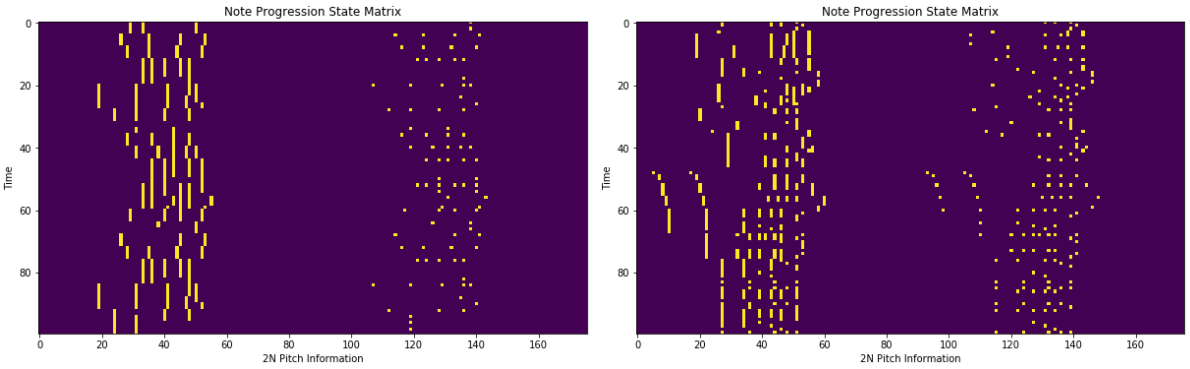
This section covers the procedure for representing the MIDI data in a binary note state-matrix suitable for NN training in Section 6.1. This is followed by a discussion on the architectural details of the two generative models implemented in Section 6.2, and finally the evaluation methodology is discussed in Section 6.3.

6.1 Midi State-Matrix Representation

The note progression state-matrix representation algorithm used in this work is adopted from [40]. It takes a standard MIDI file as input and transforms the "note on" and "note off" messages into a matrix of binary entries. In this representation, only information pertaining to note messages is kept, that is, whether a note is on or off, the timing and duration of the note. The algorithm consists of two separate processes, one for encoding a MIDI file into the note state-matrix representation, the other for decoding and transforming an existing note state-matrix into a valid MIDI file that can be transcribed by any MIDI enabled device. The encoding and decoding processes are discussed below.

6.1.1 Encoding

Starting with a MIDI file with textual and numerical data, the aim is to extract and transform into binary all note information to be used as training data for music gen-



(a)

Figure 6.1: *Visualization of sample MIDI files in the note state-matrix representation.*

eration. For each file, all messages relating to the same pitch are first lined up in ascending order of their delta times as defined in Section 5.2.4, with each present pitch having a list that starts at tick time 0. Adding up all the delta times on the pitch that plays last in the song produces the duration of the song. From the assembly of messages per pitch present in a song, three important attributes, namely pitch, time and note message type, are extracted and used to create the state-matrix representation of the song.

Extracting the pitch and note message type are fairly straightforward orparations since they are explicitly contained in each of the ordered messages. The $M \times 2N$ matrix is constructed so that the M rows represent tick times. Of the $2N$ columns representing the pitch information for N possible pitch values, the first N columns uniquely identify each pitch value, and the next N indicate whether the pitch was played in the previous time step or not. N represents the number of possible pitches which is 128 by default. The entries into the matrix are binary to indicate the state of the note at each tick time on all pitches. Figure 6.1 is a binary heat map of the state matrix representation. This representation allows for easy expression of multiple chords, hence the models will be able to learn polyphonic note progressions. One complication with the state matrix representation is that it creates an imbalance in the prediction space. This is because each time-step in the matrix contains a very small proportion of note-on signals as compared to note-off signals for the 128 available pitches. Table 6.1 shows the proportion of prediction instances representing positive

note activations and negative activations for the training dataset used in this study.

Note message type	Number of note messages	Percentage
Note-off	5 924 916	96,63%
Note-on	199 884	3,37%
Total	**6 124 800**	**100%**

Table 6.1: *Number of prediction instances/note messages in the training dataset showing the data class imbalance inherent in the state-matrix representation used in this work.*

Owing to the imbalance shown in Table 6.1, a model that simply predicts note-off messages for all 128 pitch values at each time step will achieve an accuracy score of approximately 96%. BAcc, which is calculated as the weighted prediction accuracy between the number of prediction classes, is not prone to the same problem as prediction accuracy, and so it is used in this work. To create the training dataset, all training and test MIDI files are passed through this process to create a collection of note progression state matrices, which results in a three dimensional matrix. This is ideal since the models explained in Section 6.2 expect a three dimensional input.

6.1.2 *Decoding*

The decoding of a note state-matrix is the reverse process of the encoding process, and as such is highly dependent on it. This process accepts a binary state-matrix of dimension defined in the encoding phase, as input, and transforms the matrix into a valid MIDI file. Without this process, it would be hard to evaluate the quality of audio samples generated by both the encoder-decoder and WGAN models discussed in Section 6.2.

To be able to generate a valid transcribable MIDI song, each observation in the state-matrix is written as a note message in the MIDI file comprising the following attributes as a minimum requirement: channel, pitch, time, and velocity as explained in Chapter 5. Note message type is also a necessary attribute for valid MIDI transcription. For playblack purpose, a meta tempo message is also sent to the file before

all other note messages. However, the models compared in this work only model pitch progression states, and so the velocity, channel and playback tempo are kept constant at 70, 1 (Acoustic Grand Piano) and 120 respectively to ensure the audio is loud enough and of standard pace. Delta time of each message is calculated based on the number of ticks between the current message and the previous message of the same pitch, scaled on the constant file tempo. In the case that an observation in the state-matrix representation contains information about more than one pitch, multiple MIDI messages are generated from this one observation with the same delta time. The time attribute $d_{t,s}$ for pitch s is extracted only after all note progression states in the matrix are determined using the formula below:

$$d_{t,s} = \frac{v_{s_t} - v_{s(t-1)}}{\tau}, \tag{6.1}$$

where v_{st} is the tick time of the current message containing instruction for state s, $v_{s(t-1)}$ is the tick time of the previous message containing information on the same state, and $\tau = 120$ represents fixed standardizing tempo.

Decoding pitch is more complex than all the other required attributes. In the note state-matrix representation of binary entries, the first $N - 1$ columns represent pitch activations and the next N to $2N - 1$ represent pitch retention of each observation. A value of 1 in the first $N - 1$ columns is recorded as a "note_on" message, and a value of 0 denotes a "note_off" message for the pitch encoded by the column. In the next N to $2N - 1$ columns, a value of 1 instructs the MIDI transcriber to activate the corresponding pitch in column in the first $N - 1$ columns, while a value of 0 releases the pitch. Activations of a pitch of constant velocity for continuous time steps until release constitute an elongated pitch press to a human listener.

When the state-matrix attributes have been extracted and written to a MIDI file, The decoding process assumes a MIDI file type 1, where all messages are written to one track.

6.2 Models

In this section, the encoder-decoder and the adversarial generative models are presented in Subsections 6.2.1 and 6.2.2 respectively. Their architectures are explored together with some of the key structural training parameters such as activation functions, number of stacked LSTM layers, and optimizers used. Since the training configurations are different, the last part in each configuration explains how the preprocessed note state-matrix data is presented to the network for training.

6.2.1 Encoder-Decoder LSTM

The encoder-decoder configuration [14], is well suited to the modelling of sequential tasks such as text translation, textual question answering, text summarization and music generation. In this configuration, two networks joint end to end with only one objective function are trained so that the first network (encoder) is fed all the training sequences and encodes what it has learned in a low dimensional vector. The decoder then learns the mapping from this encoded "thought vector" to the desired sequence of outputs. The decoder does this by optimizing a loss function, and is trained using BPTT explained in Section 3.2.1. Since the two networks are connected, gradients from the decoder are propagated all the way back to the encoder, so the encoder also improves its encoding process.

It has been shown that stacking LSTM cells results in improved performance [37, 39], and so three stacked LSTM cells in both the encoder and decoder networks were implemented. The reason for stacking LSTM layers is to capture multiple levels of abstraction that could be inherent in the temporal data. For music generation, the sequence of note information encodes not just note progressions per track, but also chord progressions and phrases that contribute to the rhythm and melody of the input and output. Capturing both the forward and backward flow of input during encoder training has shown to significantly improve the cell state's memory retention and subsequently the quality of generated sequences [30, 39] by the decoder. For this reason, one of the encoder layers will read the input sequence backwards, the other forward, and the third takes the aggregation function $a(h_1, h_2)$ result of the other two layers's hidden states as input to finally produce the final hidden state $V_T = \theta_3(a(\vec{h}, \vec{h}))$, where

$\theta_i(.)$ comprises all LSTM equations in Section 3.2.2. Figure 6.2 shows bidirectional stacked encoder network. This bidirectional input approach was succesfully implemented for music generation in [37] and [38].

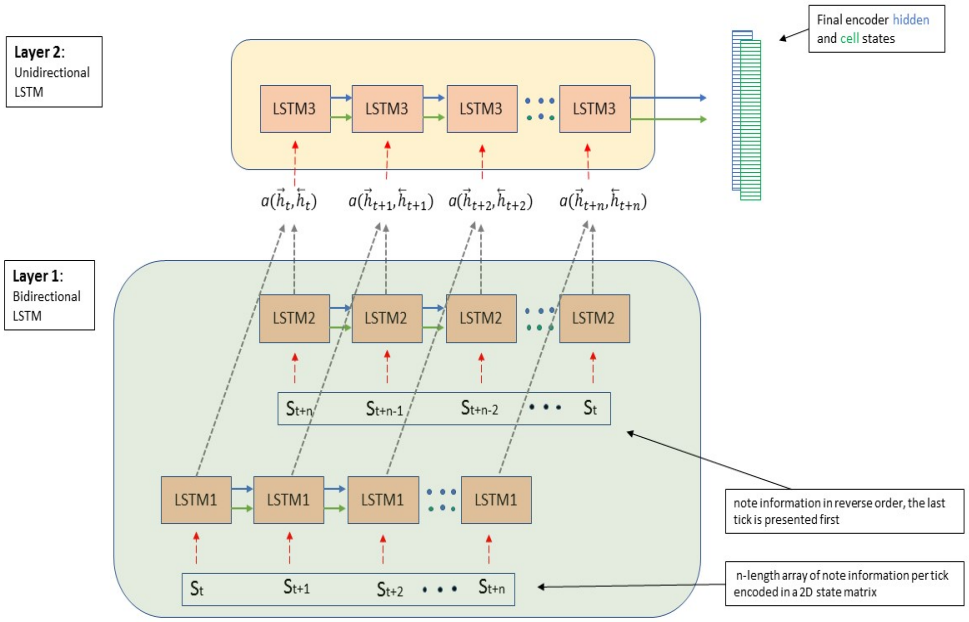


Figure 6.2: *The encoder bidirectional LSTM network. Inputs S_{t+i} represent note pitch information per timeseries observation (tick) pulled from the 2D note progression state-matrix*

The first two LSTM layers can be considered to be on the same hierarchical level, hence making up just a single bidirectional layer. The third LSTM is stacked over the bidirectional layer. Aggregation function a can either be a summation, multiplication or concatenation operation over the hidden states of the forward and backward reading LSTM cells. Each LSTM cell internally consists of standard LSTM operations.

The decoder comprises of three stacked unidirectional LSTM cells with 256 neurons each, and sigmoid output activation. Unlike the encoder, the decoder is used in two modes, notably: training and inference. These two modes differ by the manner in which information flows from input to output until termination of decoding. Dur-

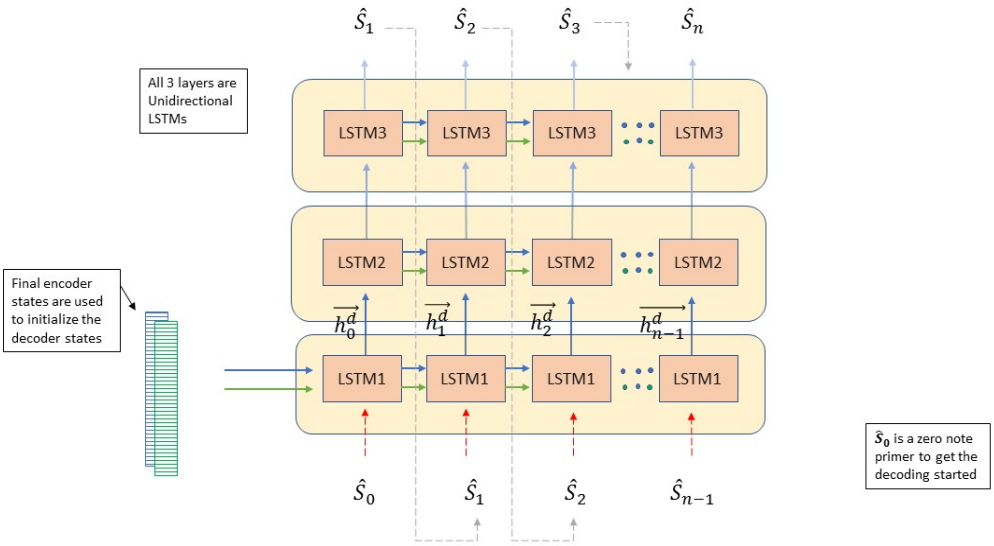


Figure 6.3: *The decoder unidirectional LSTM network.Inputs S_{t+i} represent note pitch information per timeseries observation (tick) pulled from the 2D note progression state-matrix.*

ing training, the decoder hidden and cell states are initialized using the encoder's final states. Since an LSTM cell expects three inputs, a zero vector $st_0 = \bar{0}$ is presented as a primer to the decoder, and the first predicted note progression state $\hat{st}_1$ is generated by the network. During training, the real musical note progression states $st_t, st_{t+1}, ..., st_{t+n-1}$ are presented as input per time step to predict the orderly sequence of note progression states $\hat{st}_{t+1}, \hat{st}_{t+2}, ..., \hat{st}_{t+n}$.

As in all other optimization based networks, the predictions $\hat{st}_{t+i}$ are compared to the actual note states st_{t+i} to calculate the loss. The RMSprop [32] optimizer is used to minimize the overall encoder-decoder network cross entropy objective given by Equation 3.2. RMSprop is the recommended choice for training RNNs according to Keras [64] as it speeds up training, and has been successsfully used in training RNNs on temporal data [27, 50, 65]. Once training is completed, a short sequence of notes is presented to the encoder which passes on a low dimensional vector representation of its final hidden state to the decoder. Inference proceeds with a primer input, first

note state prediction is passed as input into the decoder in the next time-step, and the process continues recursively until the desired number of time-steps of output is reached as depicted in Figure 6.3. In all the LSTM cells, dropout, discussed in Section 3.1.1.2, is used for regularization. Since each song is represented by a $T \times 2N$, where T represents number of ticks/time-steps and N the number of allowable pitches to model, all LSTM cells have layers containing $2N$ hidden units. The algorithm used for extracting note information from the MIDI file and create the note state-matrix representation was discussed in Section 6.1.1.

Encoder-decoder networks are sequence-to-sequence models, and so accept a sequence of priming notes in order to be able to produce an output sequence. In music generation, as in any other sequence-to-sequence task, the more data presented to the encoder network, the better the quality of information available to the decoder network. However, longer sequences can have negative effects on learning due to vanishing gradients. For encoder-decoder music generation, all songs are limited to N ticks; m of these, where $m < N$, are presented to the encoder and n, with $m < n < N$, to the decoder during training. Note that $m + n = N$. All training songs are split in this manner and presented to the network in batches of eight. During inference, the network expects a primer sequence of m notes and generates n note progression states probabilities, and a threshold value of 0.5 is used to turn notes on. The threshold is set to 0.5 because the values being predicted are probabilities of a note being played.

Once there is a sequence of notes generated, it is presented to the MIDI note state-matrix decoding method in Section 6.1.2 to produce a MIDI file that can play on any MIDI enabled device. In the following section, the WGAN implementation is discussed.

6.2.2 *LSTM WGAN*

As mentioned in Section 4.2.1, WGAN is an improvement over standard GAN training. The WGAN implementation in this work consists of an LSTM generator and discriminator. Since the goal is to compare adversarial training to encoder-decoder configured training, both the generator and discriminator network architectures are identical to the decoder and encoder respectively in terms of the number of LSTM

layers, number of neurons in each layer, and activation functions except for the output activation. Other training hyper-parameters such as learning rate and stopping criteria are left to vary per training configuration, and will be discussed in Chapter 7. Below is a detailed description of the WGAN implementation used in this study.

The generator $G(z)$ in WGAN is similar to any other generative model that generates melodies from random noise. The specific implementation in this study has three stacked unidirectional LSTM layers with 256 neurons in each layer, an input layer, a fully connected layer before the output, and an output layer. The input layer accepts a matrix of latent variables of size 256 for each generation time-step from a the standard normal distribution with a mean of zero, and a standard deviation of one. The first LSTM layer takes this latent matrix as input per time-step and passes the information on to the higher layers, and the state output per time-step is passed on to the next time-step to generate a sequence of outputs. The outputs should transform into realistic note state-matrix probabilities over the course of training. The output of the generator network is forced to have the same dimensions as those of the state-matrix representation of the real data.

In traditional GAN training, the discriminator $D(x)$ is an adversary to $G(z)$ as they are trained in competition. The $D(G(z))$ in WGAN acts more in partnership with $G(z)$ in that, $D(G(z))$ is trained to optimality much faster than $G(z)$, and so $D(G(z))$ is able to provide important loss information to $G(z)$ very early in training to speed up $G(z)$'s convergence. In this manner it is more comparable to the encoder-decoder network in encoder-decoder models explained in Section 6.2.1 in that it provides the necessary information for the generator's optimal training. The $D(G(z))$ implementation has the same number of LSTM layers, including one bidirectional layer as the encoder, discussed in Section 6.2.1. The network expects input with dimensions consistent to the note state-matrix representation of the MIDI files, and produces a scalar output to ensure the distance between real and fake sample outputs is as large as possible. This is unlike traditional GANs that produce a probability that the song is a sample from the real data generating distribution. For this purpose a linear output activation function is used for $D(G(z))$. The scalar output helps in calculating the EM distance defined in Section 4.2.1, which is used in $D(G(z))'s$ and $G(z)'s$ loss functions.

The discriminator takes both real note samples and random noise as inputs together with their labels; 1 for real and -1 for fake samples.The random noise is passed through a partially trained generator at that point in time for each epoch, and the resulting samples represent generated music. To ensure $D(G(z))$ is always more informed than $G(z)$ to be able to guide $G(z)$'s loss to optimality, $D(G(z))$ is trained five times more for every single epoch of $G(z)$. Figure 6.4 shows the structural setup of the WGAN configuration with $D(G(z))$ and $G(z)$ as explained above applied on image generation.

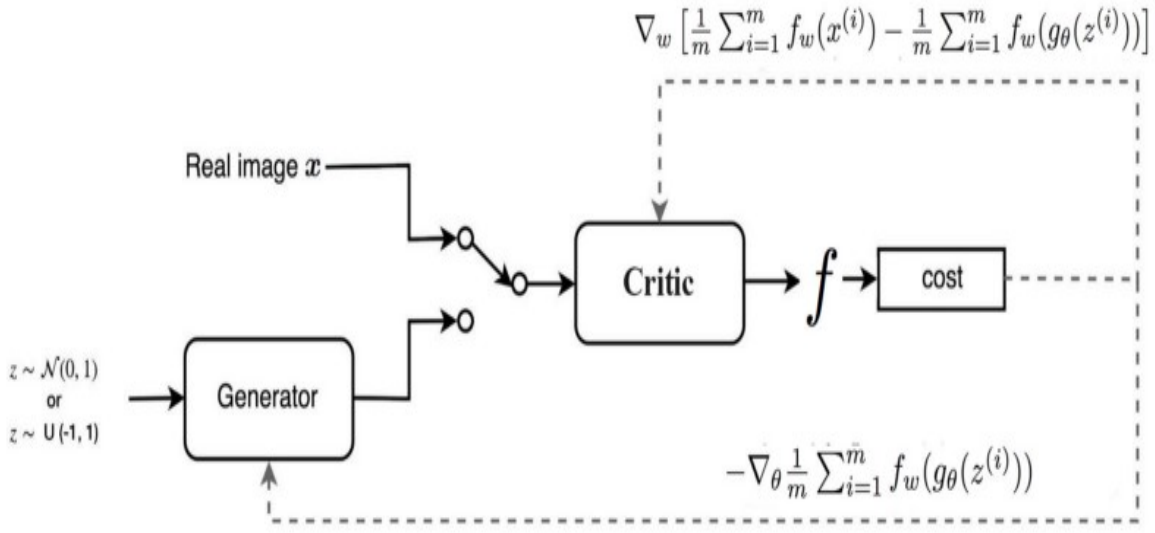


Figure 6.4: *WGAN architecture for image generation. Source: [68]*

6.3 Evaluation Methods

In machine learning, it is common to measure the training and validation accuracy of a model and use these for model evaluation. In this work, the training and validation errors of the models were recorded and will be reported in Chapter 7, however they are not a good measure for the performance of a generative algorithm. In generative models, the goal is to be able to learn the generative distribution underlying the samples used during training, while at the same time not overfitting on these few examples. This means that it is possible to have a model that has average training

and validation accuracies produce much more realistic and creative musical samples than one with a very high training accuracy, as music is an art form and is very subjective. Majority of existing literature in neural music generation relies on subjective evaluation methods, with human listener surveys being the most common approach [19,37,55,67]. For this study, a survey was conducted where listener impression scores from 10 individual volunteers were collected. The scores give a subjective opinion of the listeners impression of the samples generated by both the encoder-decoder and the WGAN generators.

The 10 human evaluators of the samples were chosen at random from a group of friends and each was contacted through Whatsapp [63] messaging to be asked to participate in the study. This method of contact was especially suitable, as it allowed for quick and easy access to individuals that would likely agree to participate in the study. The fact that the social messaging application has functionality for sharing audio was also one influential factor for the choice of survey design. All the volunteers were aged between 20 and 40 with an uneven split of gender (6 male, 4 female). They were notified that they will receive eight distinct pairs of audio samples not longer than 2 minutes in length that are both generated by a learning algorithm. They were then asked to rate each of the samples by giving it a score in the range [0, 5], (where 0 is completely random noise and 5 is a good song), to express how much they enjoyed listening to the song and how rhythmic and melodic the music samples are. A numerical score was collected for each sample in the comparison pair. This enabled calculation as a percentage of the number of times samples from one model are preferred over those from the competing model. This together with the median represent a good measure of centrality of the scores to ensure the analysis is not influenced by outliers.

Each participant received a group of eight randomized pairings of encoder-decoder and WGAN samples to compare and score, resulting in a total of 80 ratings. Microsoft Excel was used to randomize the samples and to assign them to the volunteers. The importance of mentioning to the volunteers that both samples were algorithmically generated was to ensure they do not score the samples by comparing them to what they define as really good music generated by expert human artists. This ensures they understand the aim of the study is to compare music samples from two gener-

ative models. The voluntees were not informed as to which samples were generated by which model. Some of the volunteers, although not asked, were able to provide textual explanation for their preference in the samples, which provided for a better comparison of the two generative models. Once the scores were collected, the mean opinion score (MOS) [62] test was conducted to get to the conclusion that answers the research questions. For each WGAN sample S_i^w and encoder-decoder sample S_j^{ed}, MOS $q(S)$ over all 10 volunteer ratings $r_k(S)$ is defined as:

$$q(S_i) = \frac{1}{10}\sum_{k}^{10} r_k(S_i), \tag{6.2}$$

where the rating $r_k(s_i)$ is the rating given to sample S_i by volunteer number k of 10 volunteers.The mean generator quality $Q(S_g)$ of a group of samples from the same generator g, is defined as the average MOS given by Equation 6.3 below:

$$Q(S_g) = \frac{1}{8}\sum_{i}^{8} q(S_i^g), \tag{6.3}$$

where S_i^g for $i = 1, 2, 3, ..., 8$ represents samples from generator g. Since there is subjectivity in the measure of quality, it is important to quantify how much variation there is in the recorded sample and model qualities using the standard deviation. This also expresses how much confidence is placed in the estimate of quality used, where a high standard deviation represents low confidence in the accuracy of the estimate, and a low deviation represents high confidence. The two standard deviation estimates for MOS and mean generator quality are expressed below respectively:

$$\sigma_{q(S_i)} = \sqrt{\frac{\sum_k^{10}(r_k(S_i) - q(S_i))^2}{10-1}}, \tag{6.4}$$

$$\sigma_{Q(S_g)} = \sqrt{\frac{\sum_i^{8}(q(S_i^g) - Q(S_g))^2}{8-1}}, \tag{6.5}$$

with $q(S_i)$ and $Q(S_g)$ given by Equations 6.2 and 6.3 respectively. Although one generator may have a higher MOS than the other, tests have to be performed to ensure the MOS estimate of quality is not negatively influenced by outliers, and that it indeed has a different and higher median opinion score. The Wilcoxon signed-rank t-test [73]

was used to perform the test of equal medians in ranked pair data, discussed in the next section.

6.3.1 Wilcoxon Signed-Rank T-Test

The Wilcoxon signed-rank t-test [73] is a statistical hypothesis test for ranked and paired data that assumes no predefined population distribution over which the data is sampled from. Wilcoxon signed-rank t-test is used for comparing related pair samples under the hypothesis that the median difference between the samples is zero.

The test makes the following assumptions about the data:

- The data observations are paired samples from the same population.
- Each pair is chosen randomly and independently.
- The observations are measured on an ordinal, not necessarily nominal scale.

The assumptions above are met by the opinion score data collection process to a suitable extent in that: (1) the ranked samples come from the same population of ranking volunteers. (2) The pairs were chosen randomly, though independence in this case is subjective as it is important to ensure all samples from both models were evaluated by the same number of volunteers. This was achieved by random selection without replacement. (3) Although the scores indicate by how much one sample is better than the other, hence violating the ordinallity assumption, a transformation is applied to the scores to ensure only the ordinal aspect of the scores is used. In this transformation, the score pairs are compared, and a new indicator feature is constructed that assumes the following values: positive (+) if the second sample has a higher score, zero if the scores are tied, and negative (-) otherwise. With this new signed transformed data, all the test assumptions are satisfied.

The null and alternative hypothesis are given by:

H_0: The median score difference between the paired samples is zero.
H_1: The median score difference is not zero.

Let *ed* and *wg* represent the encoder-decoder and WGAN models respectively and *sgn* represent the sign function. With data observations $sgn(r_{ed,i} - r_{wg,i})$, all tied pairs are excluded from the initial sample of size N to a reduced test sample of size N_r. The original sample pairs are ordered in ascension according to the absolute differences $r_{ed,i} - r_{wg,i}$ of the original captured scores with the smallest difference ranking first as 1 and all ties receiving the average rank of the positions they span. With these new pair ranks R_i, the Wilcoxon signed-rank t-test statistic is calculated as follows:

$$W = \sum_{i}^{N_r} (sgn(r_{ed,i} - r_{wg,i}) \cdot R_i). \tag{6.6}$$

Note that using the reduced sample size N_r is equivalent to using the original sample size N, since all tied pairs will result in a zero sign hence not contributing to W. For $N_r \geq 10$, W is asymptotically normally distributed, thus the z-score can be calculated as follows:

$$z = \frac{W - 0.5}{\sigma_W}, \tag{6.7}$$

with:

$$\sigma_W = \sqrt{\frac{N_r(N_r + 1)(2N_r + 1)}{6}}. \tag{6.8}$$

The null hypothesis H_0: equal medians is rejected in favour of H_1: unequal medians, if $z > z_{critical}$. Rejecting the null hypothesis would mean the two models have generated sample scores with statistically different medians, and so there is more confidence that the contribution of outliers in the mean model and opinion scores is trivial.

Based on the results of the MOS's, mean generator quality, and the Wilcoxon signed-rank test, the generator with the higher mean generator quality will be considered to produce music that is more aesthetically pleasing to listen to, given the null hypothesis of equal medians is rejected. The volatility estimates given by Equation 6.5 are used as a measure of the confidence in making the conclusion based on the MOS estimates above. The textual comments collected on some samples will be analysed to get more insight into how people perceived the music. However, due to the lack of correct collection of these comments, no computational sentiment analysis will be

done on the data, but a human opinion sentiment analysis of the text will be provided.

6.4 Conclusion

This chapter discusses the two generative models being compared in this study, namely the encoder-decoder and WGAN. The chapter concludes by explaining the methods of evaluation to be used in getting to a conclusion for the study. In the next chapter, the training and evaluation experiments are discussed.

Chapter 7

Experiments

In Chapters 5 and 6, the data and all implementation methods were explained. In this chapter, the experimental processes followed to produce the results that address the main questions of this study are presented. In Section 7.1, implementation configurations of the MIDI state-matrix representation are briefly discussed, followed by a discussion on the training parameters for both generative architectures in Section 7.2. Finally, Section 7.3 is focused on the generation of music samples used in evaluating the encoder-decoder and WGAN models. All computation relating to the data and models was performed on a 7th generation core I5 intel processor, two Gigabytes Nvidia GeFORCE GPU personal computer with 8 Gigabytes of RAM and a Terabyte of ROM.

7.1 MIDI Representation

Of the 128 possible pitch values, existing implementations use only 88 pitch values between 21 and 109 since all other pitches outside this range are inaudible to the human ear. This in effect reduces the state-matrix dimensionality, and overall model complexity. In this work, all 128 pitch values are used to avoid the added pre-and-post processing in using a reduced note representation. For the models trained below, changing from 88 pitch values to 128 pitch values had an increase in model parameters of only 16% and 22% for the WGAN and the encoder-decoder models, respectively.

7.2 Model Hyper-Parameters

Below is the final list of models training hyper-parameters used in the study. A brief explanation of how they affected learning then follows per model.

7.2.1 Encoder-Decocer

The training hyper-parameters for the encoder-decoder LSTM neural network are shown in Table 7.1.

Hyper-parameter	**Value**
Learning rate	0.001
Optimizer	RMSprop
Dropout Probability	0.2
Training epochs	300
Batch size	5
Hidden layer size	256
Gradient clipping max	2.0
Train test split	80:20

Table 7.1: *Training hyper-parameters for the encoder-decoder neural network.*

	Batch Size		
Learning Rate	**5**	**8**	**10**
0.0005	71.2%	56.6%	51.1%
0.001	**74.1%**	69.6%	60.3%
0.005	73.6%	73.2%	72.8%

Table 7.2: *Mean 5-fold CV balanced accuracy scores.*

The learning rate was set low to ensure smoother tracking down the loss function as higher learning rates tend to converge quicker, but the quality of music generated was

bad hinting at convergence to a local minimum. Dropout [10] as a regularization technique added more training stability and reduced overfitting. Other parameters such as gradient clipping and batch size were determined using five-fold cross validation with all other parameters fixed. Table 7.2 shows mean BAcc scores over the five-fold cross validation (CV) for different learning rates and batch sizes. The model's training mini-batch size and learning rate were closen from the 5 fold CV parameter search shown in Table 7.2, and they are the values that achieved the highest BAcc score.

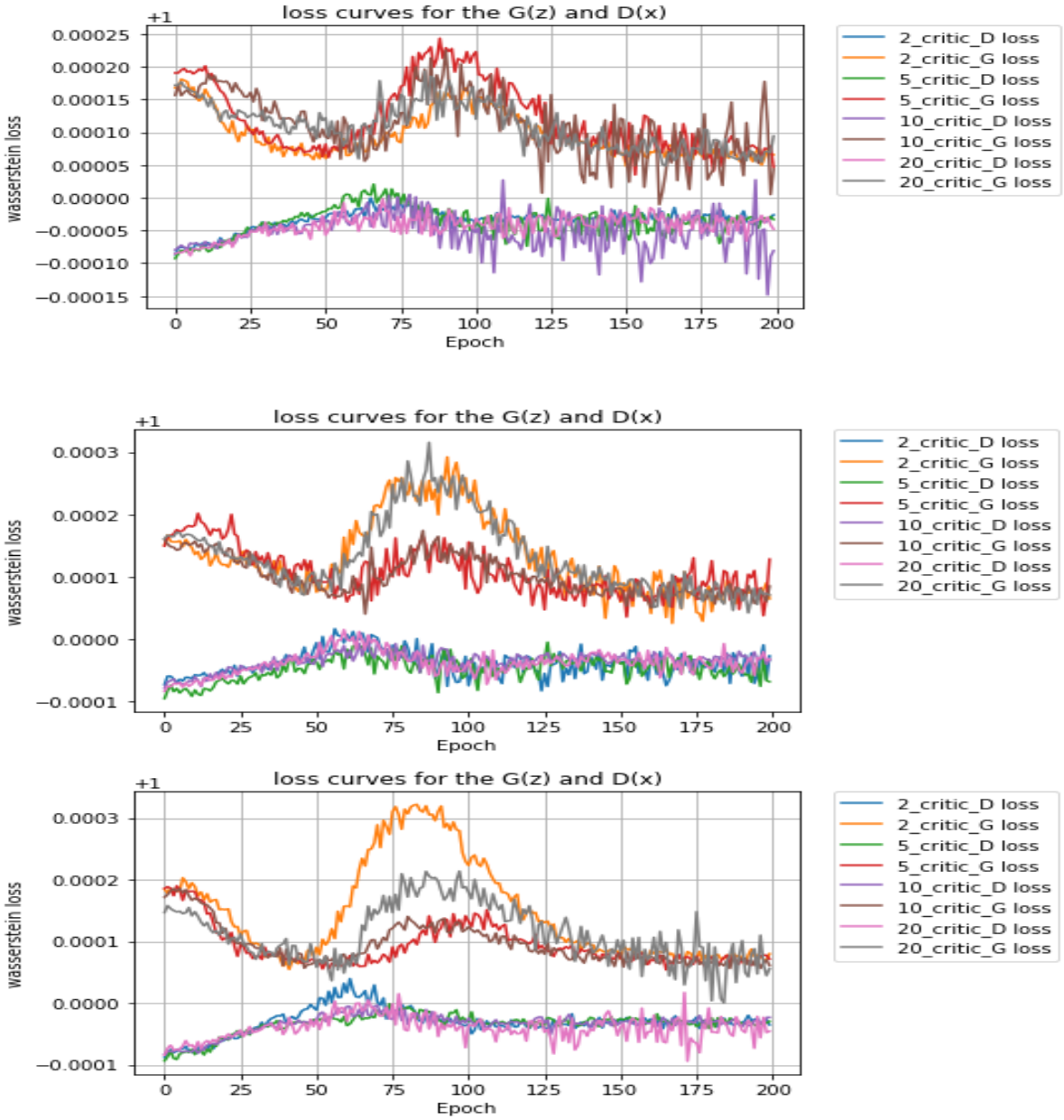
7.2.2 *WGAN*

Table 7.3 shows the final training hyper-parameters for both the generator and discriminator LSTM networks of the WGAN.

Hyper-parameter	**Value**
$G(z)$ learning rate	0.00005
$G(z)$ optimizer	RMSprop
$G(z)$ epochs	4000
$G(z)$ batch size	32
z distribution	Standard normal
$D(x)$ learning rate	0.00005
$D(x)$ optimizer	RMSprop
$D(x)$: $G(z)$ epoch ratio	5:1
Gradient clipping	0.01

Table 7.3: *Training hyper-parameters for the WGAN model.*

Training the WGAN has more moving parts than the encoder-decoder model. The initial training parameters were adopted from Gulrajani et al [44], and then a grid search was used to fine tune the hyper-parameters over 200 epochs. Figure 7.1 shows loss curves for both $G(z)$ and $D(z)$ for different points in the hyper-parameter grid search space. The combination of parameters that led to stable training were chosen as the final training parameters shown in Table 7.3.



(a)

Figure 7.1: *The figure above shows $G(z)$'s and $D(x)$'s training losses for 12 grid search points, where the number of critic training epochs (n_critic) and batch size are varied.* **Top**: *Loss curves for batch size = 32. for n_critic = 2, 5, 10 and 20* **Middle**: *Loss curves for batch size = 64.* **Bottom**: *Loss curves for batch size = 128.*

At the start of training the generator takes Gaussian noise together with the discriminator's critic of the generated samples. During this initial training the discriminator's layers are fixed and not trained to ensure the generator learning has started by the time the discriminator is trained to optimality. For each epoch of the generator, the discriminator is trained for five times more epochs, then it is used to critic the generator at its current competence level.

Since the RMSprop optimizer [32] is used, the learning rates for both the generator and discriminator decay with the number of epochs. This is to force the generator to value learned signal more than temporary noise caused by sudden inconsistent gradient direction changes. Training of the WGAN model was faster than that of the encoder-decoder model and had fewer combined generator and discriminator trainable weights. This is because there is no transfer of LSTM cell and hidden states between the generator and discriminator unlike there is between the encoder and decoder networks. The WGAN networks are connected only by a scalar loss for information transfer. The generator and discriminator are not trained to optimize accuracy since the aim is not regenerating any song from the training set, and so the losses are rather similarly to the MSE loss. The early stopping criteria for training was set to be when the EM distance between the current generator's learned distribution and that of the real data stops reducing for any 10 consecutive epochs.

7.3 Music Generation

For the encoder-decoder model, music is generated one sample at a time at the end of training by priming the decoder network with a short series of notes and predicting the note progression probabilities for the entire song. The generated samples are on average 1.3 minutes long. It turned out that generating melodies that are much longer than 2 minutes resulted in repetitions of the same sequence of notes and sometimes silent spots or complete silence towards the end of the song.

For WGAN, samples are generated during training. At the end of each generator training epoch, a sequence of ten priming latent vectors are passed to the generator, and it then generates ten audio samples based on its proficiency at that point in time.

This process is repeated until training is complete, resulting in a number of music samples. If training progresses as expected, the samples reflect an increase in compositional skill from epoch one to the last epoch. Only randomly selected samples from the last generator training were collected and written to a MIDI file for evaluation.

7.4 Conclusion

This chapter covered the experimental set up, starting with MIDI data representation, followed by a discussion on how the encoder-decoder and WGAN model architectures and hyper-parameters were set. The process of producing music samples from both trained models was then discussed in Section 7.3. In the next chapter, the experimental results are presented and analysed.

Chapter 8

Results and Analysis

The previous section presented experiments on training the two models together with the music composition process and evaluation of music samples from the trained encoder-decoder and GAN models. Here, the results from these experiments are discussed. The training, validation and test errors are discussed in Section 8.1, followed by reporting and analysis of the mean opinion scores of samples from the two models in section 8.2. Section 8.3 covers the Wilcoxon signed-rank t-test used in comparing sample medians, and finally, listener impression comments are presented in Section 8.4.

8.1 Training and Test Results

A model's predictive accuracy is a quantitative measure of the number of prediction instances the model estimated correctly divided by the total number of prediction instances as a percentage. In the case of musical notes, accuracy is measured as mean number of notes correctly classified as on or off per time step. The accuracy is measured during both training and validation of the model to ensure it is not over-fitting and is generalizing well. This is measured on the validation set. The table below shows the loss and accuracy scores for the encoder-decoder model together with the WGAN's EM distance loss, which like the MSE loss, has only a lower bound of zero and no upper bound.

	WGAN LSTM	Encoder-Decoder LSTM
Training Accuracy	-	96,45%
Test Accuracy	-	96,3%
Training entropy loss	-	0.107
Test entropy loss	-	0.081
Generator loss	1.0006	-
Discriminator loss	0.9995	-

Table 8.1: *Training and test accuracies.*

CV Iteration	Training BAcc	Validation BAcc	Training loss	Validation loss
1	74.16%	73.84%	0.64%	0.78%
2	74.27%	74.66%	0.65%	0.30%
3	74.75%	74.78%	0.29%	0.16%
4	74.34%	74.24%	0.61%	0.67%
5	74.61%	74.09%	0.31%	0.46%
μ	**74.42%**	**74.32%**	**0.5%**	**0.48%**
σ	**0.24%**	**0.39%**	**0.18%**	**0.25%**

Table 8.2: *Training and five-fold CV balanced accuracy scores for the encoder-decoder LSTM neural network.*

Although the prediction accuracy reported in Table 8.1 seem favourably high, they are a bad measure of the generative ability of the encoder-decoder model on the training data. This is because there are more negative than positive prediction instances in the dataset as described in Section 6.1.1, and shown in Table 6.1. To solve this problem, BAcc that results in a weighted score between the number of prediction classes was used. Since the models trained depend on random initialization of weights, 5-fold CV was performed to get the average performance of the models. Table 8.2 shows the CV BAcc and loss that are a more realistic measure of the model's prediction ability.

Results from Tables 8.1, 8.2 and 8.3 are not sufficient to arrive at a conclusion in comparing WGAN to the encoder-decoder model for the purpose of music generation

CV Iteration	$G(z)$ **loss**	$D(z)$ **loss**
1	1.000587	999953
2	1.000384	0.99962
3	1.000457	0.99959
4	1.000535	0.99956
5	1.000561	0.99952
μ	**1.0005**	**0,9996**
σ	**0.00008**	**0.00017**

Table 8.3: *Training and five-fold CV EM loss for the WGAN model.*

since they follow very different training methods with completely different objective functions. Also, music quality is more subjective as an art form than it is objective, and so there is a lack of good objective measures to base a conclusion on [37, 55, 60].

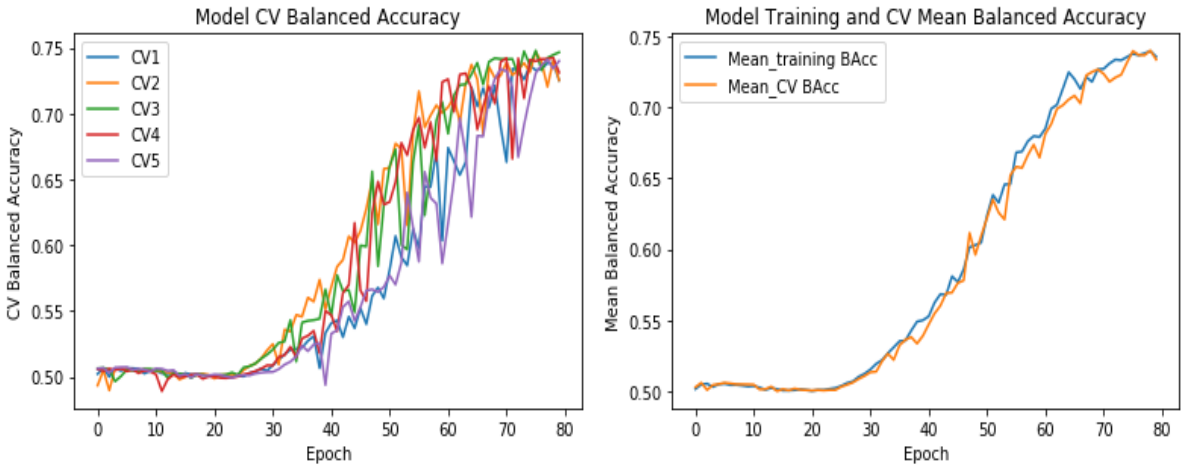


Figure 8.1: Left: *Five-fold CV BAcc curves for the encoder-decoder LSTM.* **Right**: *Mean training and CV BAcc curves.*

Accuracy reported for the encoder-decoder model represents the mean fraction of correct pitch predictions made by the decoder in the training song per prediction class, and is depicted in the training curves shown in Figure 8.1. The EM distance for WGAN is the final distance of the generators learned sampling distribution from that of the training data distribution.

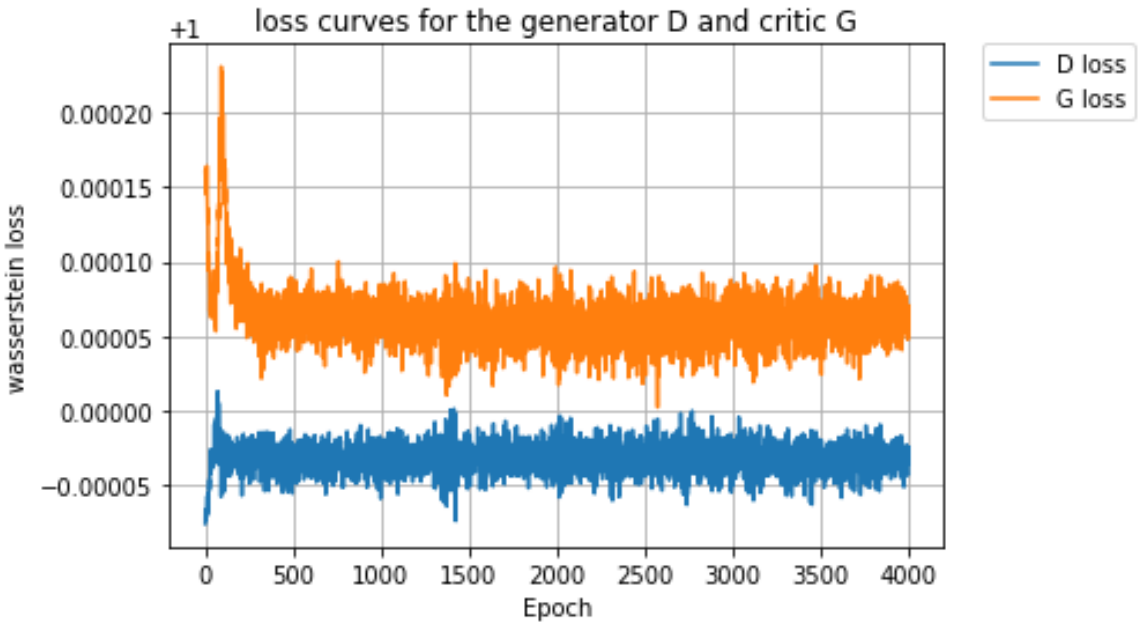


Figure 8.2: *Loss curves for WGAN with the best grid search parameters: n_critic=5 and batch size=32.*

The close training and CV mean BAcc curves in Figure 8.1 also show the network isn't overfitting. An overfitting model would regenerate the training music data samples when primed with a close enough sequence of notes and would lack diversity in the generated samples. Although the encoder-decoder does not seem to be overfitting based on the CV accuracy, volunteers in the listening survey do hint at lack of diversity and creativity in the encoder-decoder's generated samples, discussed in Section 8.4.

Figure 8.2 contains training EM distance loss curves for the WGAN networks. Note that the generator is much more unstable than the discriminator. The generator produces an estimate of the true distribution, and the discriminator's loss estimates how far off the generator is through the EM distance. As stated earlier, due to the subjective nature of music, a subjective evaluation method was used for the generated samples and the results are presented in Section 8.2.

8.2 Mean Opinion Scores

In this section, the results of the MOS survey for the two generative models are presented.

Volunteer	S1	S2	S3	S4	S5	S6	S7	S8
vol1	2,5	4	2	3	2,5	4	4	3
vol2	4	3,6	4	3	3	2	3,5	2,5
vol3	3	3	3	4	3	3,5	4	3
vol4	4	4	2	3	3,5	4	3	2
vol5	3,5	4	4	4	3	2,5	3	4
vol6	3	1	2	4	2	3	3,5	2,5
vol7	2,8	4	5	3,5	1	5	3	3
vol8	2,5	4	3,5	3	3	4	4	3
vol9	1,5	4	2,5	3	4	4	3	4,5
vol10	2	4	2,5	4	3	3	3,5	2,5
$q(s_i)$	**2,88**	**3,56**	**3,05**	**3,45**	**2,88**	**3,5**	**3,45**	**3**
$\sigma_{q(s_i)}$	**0,77**	**0.91**	**0,99**	**0,47**	**0,78**	**0,84**	**0,42**	**0,71**

Table 8.4: *Listener impression scores for WGAN generated samples S1 to S8. $q(s_i)$ represents the MOS for each sample.*

The results in Tables 8.4 and 8.5 show that volunteers generally scored WGAN samples higher than the encoder-decoder samples. This is reflected in the observation that only one of four WGAN samples received a MOS below three, yet all encoder-decoder samples got a MOS of three or lower. All samples from both models had score variations that are considerably low (all below one), also, both models have 95% sample ratings that are within two standard deviations of each other. This is a good indication that as much as the rating system is subjective and high variance is expected, people's opinions about the samples do not differ so significantly that it were to seem they are each receiving a song of a different genre or were all exposed to entirely different interpretations of what good music sounds like.

The overall results for both models across all samples are presented in Table 8.6, and

Volunteer	S9	S10	S11	S12	S13	S14	S15	S16
vol1	3	3,3	1	2	2	3	2,5	3,5
vol2	2	3	3,5	2	3	3	4	2
vol3	2	4	2,5	2	2	3,5	3	2
vol4	4,5	3	3	5	2	3	2,5	3
vol5	2	2	3	2	1	3	2	2
vol6	4	2	3	3	3,5	2,5	4	3,5
vol7	2,5	3	4	2	3	4	2	4
vol8	3,5	3	3	3,5	2	3,5	3	2
vol9	3	2	3,5	2	3	3	3	3
vol10	3,5	3	3,5	3	3	4	2	4
$q(s_i)$	**3**	**2,83**	**3**	**2,65**	**2,45**	**3,25**	**2,8**	**2,9**
$\sigma_{q(s_i)}$	**0,84**	**0,62**	**0,77**	**0,95**	**0,72**	**0,46**	**0,71**	**0,8**

Table 8.5: *Listener impression scores for the encoder-decoder LSTM generated samples S9 to S16.*

	WGAN LSTM	Encoder-Decoder LSTM
Q_{q_s}	3,21	2,86
$\sigma_{Q(q_s)}$	0,811	0,736
$Q_{q_s} + 2 \times \sigma_{Q(q_s)}$	4,832	4,332
$Q_{q_s} - 2 \times \sigma_{Q(q_s)}$	1,588	1,388
Median Score	3	3

Table 8.6: *Mean and Median generator quality scores. Although the sample median scores from the two models are equal, this does not imply they are drawn from polulations with equal median scores. It is the Wilcoxon signed-rank t-test that gives a conclusive answer on equality of the population medians.*

will be used to deduce which of the two models is considered a more skilled composer of music than the other based on its generated samples. The results in Table 8.6 show that the WGAN samples have a higher mean opinion score than those of the encoder-decoder model based on volunteer listener's ratings. However, since the arithmetic mean is a measure that can be greatly influenced by outliers, it is also important to consider the median opinion scores which is not influenced by outliers and is a measure of centrality of data observations. The Wilcoxon signed-rank test is used to assess whether the opinion scores collected for the WGAN and encoder-decoder

models come from populations with equal medians. The Wilcoxon test results are discussed in the section below.

8.3 Wilcoxon Signed-Rank Test

The Wilcoxon signed-rank test for equal population medians as explained in Section 6.3.1 was performed using a Microsoft Excel workbook as it has built-in square root functions, normal distribution, and many more functions. Using Equations 6.6, 6.7 and 6.8 as described in Section 6.3.1, the test statistic W, signed-rank standard deviation σ_W and the z-score z were calculated and are presented In Table 8.7. The tabulated results are used in making the decision to reject, or to not reject the null hypothesis of equal population medians.

W	σ_W	$\lvert z \rvert$	$z_{\alpha=0,95}$
-1132	393,90	2,8751	1,6449

Table 8.7: *Wilcoxon signed-rank test results*

It is standard to perform hypothesis tests to a certain level of confidence, usually $\alpha = 0,95$. The critical $z_{0,95}$ value is the inverse standard normal value under which 95% of all data will fall since the standardized signed-rank test statistic z follows a standard normal distribution.

Results in Table 8.7 show that z $>$ $z_{0,95}$ and according to the Wilcoxon signed-rank test, the null hypothesis of equal medians is rejected with 95% confidence in favour of the alternative hypothesis. The alternative hypothesis states the two generative models produced samples with significantly different opi-nion score medians and thus their population distributions are not centered around the same score. This is also supported by plotting a histogram of opinion scores for both models in Figure 8.3.

Figure 8.3 shows that WGAN sample scores are skewed more to the right as compared to those of the encoder-decoder network that form a more symmetric distribution. The evident skewed WGAN sample score distribution and a higher median opinion score suggest the higher WGAN MOS score is not falsely influenced by outliers, and

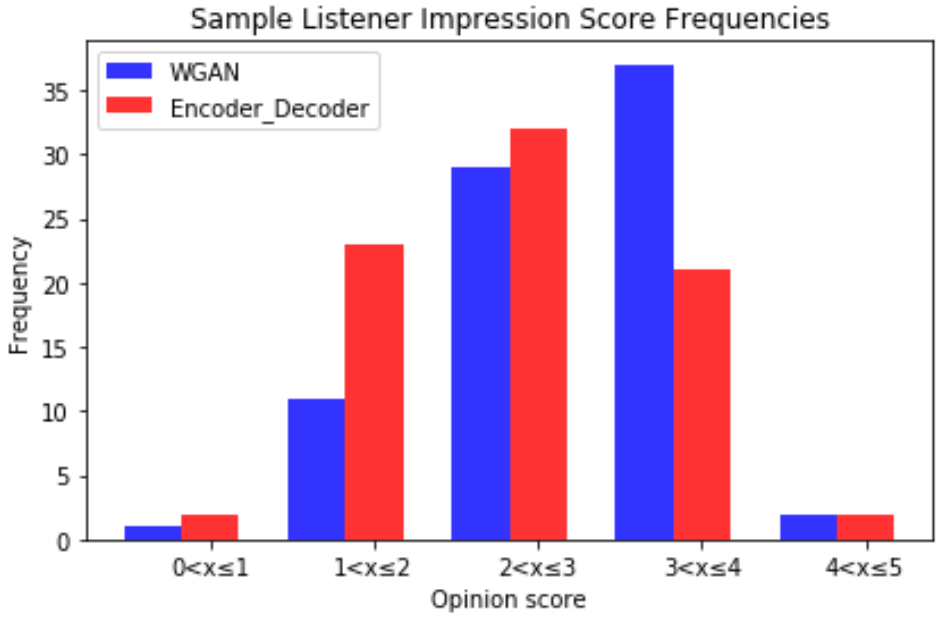


Figure 8.3: *Opinion score distribution for WGAN and encoder-decoder LSTM generated music samples.*

this too supports that WGAN samples were found to be more pleasing to listen to than those generated by the encoder-decoder model.

8.4 Listener Comments

Although volunteers were not asked to provide any textual comments, some of them did provide written feedback together with the rating scores. Majority of the comments provided justification for the volunteer's rating of the samples and their preference for one sample over the other. The generating model identity of the samples were not known to the volunteers during the survey, named as simply sample 1 to 8 for the WGAN, and 9 to 16 for the encoder-decoder model. In total, 26 of the ratings were accompanied by a comment. Some of the most insightful comments are provided in the list below, with the generating model's name instead of the sample number revealed.

- *"I couldn't get a feel of where the encoder-decoder song is going, the WGAN sample has a nice classical feel to it."*
- *"They both sound musical but the sound quality is bad."*
- *"I like the pace of the WGAN sample."*
- *"The encoder-decoder sample had too many silent spaces, the quiet spots emphasize the loud spots ,sounded like rambling."*
- *"The WGAN sample is a little chaotic but generally creates a good atmosphere, the encoder-decoder song has good rhythm but no actual melody.*
- *"I'd say the encoder-decoder song is better, has more rhythm, the WGAN sample sounds too fast and just noise to me."*
- *"The WGAN Sample sounds like me when I am under dissertation stress, terrible."*
- *"The WGAN sample sounds more creative, the encoder-decoder sample wins on rhythm although it takes soo long to get there."*
- *"What's important to me is they all sound musical."*
- *"I don't listen to this genre so my view may be way off."*
- *"Most of the songs sond similar to me."*
- *"Wow, can't believe the WGAN sample was generated by a machine, though it's funny at the end LOL."*
- *"Can't you get them to generate longer songs? perhaps with words? I like the encoder-decoder sample."*

The comments point to a general preference for WGAN music samples over those of the encoder-decoder model. WGAN sample comments can be summarized using the following keywords: *melodic, rhythmic, nice, creative*. The encoder-decoder samples can be described: *Slow, vague, rhythmic*.

Chapter 9

Conclusion

This dissertation described and compared two training configurations of generative models for music generation using MIDI data. After a detailed discussion of neural network architectures normally used for time series data and their topological components, state of the art music composition networks were discussed. A short background on the MIDI file format and its representation in this work was provided. The work then detailed the WGAN and encoder-decoder models with LSTM components implemented and compared for the composition task, followed by the experimental setup and results.

The main purpose of the study was to first show that the adversarial configuration is a viable approach to training neural networks for music composition, and that it produces music samples of superior quality as compared to non-adversarial training. To be able to evaluate the generative skill of the two compared training configurations, a survey was conducted where 10 volunteers were asked to listen to four randomly selected samples from each generative model, and rate each sample on a scale from 1 to 5 based on how pleasing it was to listen to. 70% of rating instances resulted in preference for WGAN over the encoder-decoder samples. 30% of volunteer ratings pointed to a preference for encoder-decoder generated samples over WGAN samples. The ratings were then used to conduct a rating test to determine whether the two model's sample ratings were from populations with different medians, and this test concluded they were. Assuming the sample ratings are representative enough of their populations, this implies adding more volunteers to the survey would infer that

the adversarial neural network is a better composer than the encoder-decoder neural network.

Based on the results of the survey and comments from volunteers on the generated samples, it can be concluded that adversarial training is a viable method for training generative models for music generation, and it does produce more diverse and pleasing to listen to music samples. Further experiments can be conducted on the WGAN implementation above by including more varied training music genres, and allowing the generator to generate longer sequences. Since the generator is an LSTM network, methods normally used to improve performance and memory retention in sequence-to-sequence models on natural language processing (NLP) tasks, such as adding an attention [25] or pointer [26] layer can be explored.

Another avenue of further research to be explored is the development of formal and objective evaluation methods for artistic tasks such as music generation and artistic painting. Being able to accurately and appropriately evaluate models for such subjective tasks will greatly improve the quality of generated samples.

Bibliography

[1] F. Rosenblatt, *"The perceptron: A probabilistic model for information storage and organization in the brain"*, Psychological Review, vol. 65, no. 6, pp. 386-408, 1958

[2] K. Arpathy, *"CS231n Convolutional Neural Networks for Visual Recognition"*, Stanford University ,\T1\textquotedbllefthttp://cs231n.github.io/neural-networks-1/\T1\textquotedblright, 2017, [online]; Accessed:23-07-2018

[3] P. Werbos, *"Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences"*, PhD thesis, Harvard University, 1974

[4] A. Moujahid, *"Data in Practice"* ,\T1\textquotedbllefthttp://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe\T1\textquotedblright, 2016, [online]; Accessed:03-06-2018

[5] S. Hochreiter, *"The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions"*, International Journal of Uncertainty Fuzziness and Knowledge Based Systems, vol. 6, no. 2, pp. 107-116, 1998

[6] P. Werbos, *"Backpropagation through time: what it does and how to do it"*, Proceedings of the Institute of Electrical and Electronics Engineers, vol. 78, no. 10, pp. 1550-1560, 1990

[7] V. Nair, G.E. Hinton, *"Rectified Linear Units Improve Restricted Boltzmann Machines"*, ICML'10 Proceedings of the 27th International Conference on Machine Learning, pp. 807-814, 2010

[8] B. Hanin, *"Universal Function Approximation by Deep Neural Nets with Bounded Width and ReLU Activations"*,arXiv:1708.02691 [stat.ML], 2017, [Online]; Accessed: 20-07-2018

[9] A. Ng, *"Feature selection, L1 vs. L2 regularization, and rotational invariance"*, Proceedings of the 21st international conference on Machine learning, pp. 78-78, 2004

[10] N. Srivastava, G.E. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, *"Dropout: A Simple Way to Prevent Neural Networks from Overfitting"*, Journal of Machine Learning Research, vol. 15, no.1, pp. 1929-1958, 2014

[11] Y. LeCun, L. Bottou, G. Orr, K. Müller, *"Efficient BackProp"*, Proceedings of the International Conference on Neural Information Processing Systems, Neural Networks: Tricks of the Trade, pp. 9-50, 1996

[12] D.P. Kingma, and J.L. Ba, *"ADAM: A method for stochastic optimization"*, International Conference on Learning Representations, 2015

[13] H. Jaeger, *"A tutorial on training recurrent neural networks"*, covering BPPT, RTRL, EKF and the "echo state network" approach. http://minds.jacobs-university.de/sites/default/files/uploads/papers/ESNTutorialRev.pdf, 2005.

[14] K. Cho, B. Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk and Y. Bengio, *"Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation"*, Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, pp. 1724–1734, 2014

[15] S. Hochreiter and J. Schmidhuber, *"Long Short-Term Memory"*, Neural Computation, Vol 9, pp. 1735-1780, MIT Press Cambridge, 1997

[16] D. J. Im, C. D. Kim, H. Jiang and R. Memisevic, *"Generating images with recurrent adversarial networks"*, arXiv preprint: https://arxiv.org/abs/1602.05110, 2016, [Online]; Accessed: 20-07-2018

[17] C. Olah, *"Understanding lstm networks"*, Blog post: http://colah.github.io/posts/2015-08-Understanding-LSTMs/, 2015, [Online]; Accessed: 20-07-2018

[18] T. Silva, *"A Short Introduction to Generative Adversarial Networks"*, Blog : https://sthalles.github.io/intro-to-gans/, 2017, [Online]; Accessed: 21-10-2018

[19] A. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior and K. Kavukcuoglu, *"WaveNet: A Generative Model For Raw Audio"*, Proceedings of the 9th International Speech Communication Association, Speech Synthesis Workshop, pp. 125-125, 2016

[20] I. Sutskever, O. Vinyals, Q. Le, *"Sequence to Sequence Learning with Neural Networks"*, Advances in neural information processing systems, vol. 4, 2014

[21] J. Briot, G. Hadjeres, FD. Pachet,*"Deep Learning Techniques for Music Generation - A Survey"*, arXiv:1709.01620: https://arxiv.org/abs/1709.01620, 2017

[22] V. Kalingeri and S. Grandhe, *"Music Generation Using Deep Learning"*, arXiv 1612.04928: https://arxiv.org/abs/1612.04928, 2016

[23] M. Alfonseca, M. Cebrian, Alfonso O. Puente, *"A simple genetic algorithm for music generation by means of algorithmic information theory"*, Proceedings of the Institute of Electrical and Electronics Engineers Congress, Evolutionary Computation, pp. 25-28, 2007

[24] A. See, P. Liu, C. Manning, *"Get To The Point: Summarization with Pointer-Generator Networks"*, Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, vol. 1, pp. 1073–1083, 2017

[25] M. Denil, L. Bazzani, H. Larochelle, and N. de Freitas , *"Learning where to Attend with Deep Architectures for Image Tracking"*, Institute of Electrical and Electronics Engineers, Neural Computation, vol. 24, no. 8, pp. 2151-2184, 2012

[26] O. Vinyals, M. Fortunato, and N. Jaitly, *"Pointer Networks"*, Proceedings of the 28th International Conference on Neural Information Processing Systems, vol. 2, pp. 2692-2700, 2015

[27] K. Lopyrev, *"Generating news headlines with recurrent neural networks"*, arXiv preprint https://arxiv.org/abs/1512.01712, 2015

[28] A. Oord, N. Kalchbrenner and K. Kavukcuoglu, *"Pixel Recurrent Neural Networks"*, Proceedings of the 33rd International Conference on Machine Learning, vol. 48, pp. 1747-1756, 2016

[29] Y. Wang, and F. Tain, *"Recurrent residual learning for sequence classifcation"*, Empirical Methods in Natural Language Processing, 2016

[30] M. Schuster, and K. Paliwal, *"Bidirectional recurrent neural networks"*, Institute of Electrical and Electronics Engineers, Signal Processing, vol. 45, no. 11, pp. 2673–2681, 1997

[31] J. F. Nash, *"Equilibrium points in n-person games"*, Proceedings of the national academy of sciences. USA vol. 36, no. 1, pp. 48–49, 1950

[32] M. C. Mukkamala, M. Hein, *"Variants of RMSProp and Adagrad with Logarithmic Regret Bounds"*, Proceedings of the 34th International Conference on Machine Learning, vol. 70, pp. 2545-2553, 2017

[33] I. Goodfellow, J. Pouget-Abadie , M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, *"Generative Adversarial Nets"*, Proceedings of the 27th International Conference on Neural Information Processing Systems, pp. 2672-2680, 2014

[34] A. Elgammal, B. Liu , M. Elhoseiny, M. Mazzone, "CAN: Creative Adversarial Networks Generating "Art" by Learning Styles and Deviating from Style Norms", Proceedings of the International Conference on Computational Creativity, 2017

[35] F. Juefei-Xu, V. N. Boddeti and M. Savvides, *"Gang of GANs:Generative Adversarial Networks with Maximum Margin Ranking"*, arXiv: https://arxiv.org/abs/1704.04865, 2017, [Online]; Accessed: 21-07-2018

[36] L. Yu, W. Zhang, J. Wang and Y. Yu, "SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient", Proceedings of the 31st AAAI Conference on Artificial Intelligence, Vol. 31, 2017

[37] O. Mogren, *"C-RNN-GAN: Continuous recurrent neural networks with adversarial training"*, Proceedings of the 30th International Conference on Neural Information Processing Systems, Constructive Machine Learning Workshop, Spain, 2016

[38] I. Liu and R. Randall, *"Predicting Missing Music Components with Bidirectional Long Short-Term Memory Neural Networks"*, Proceedings of the 17th International Society on Music Information Retrieval Conference, New York, 2016

[39] Z. Cui, R. Ke, Y. Wang, *"Deep Stacked Bidirectional and Unidirectional LSTM Recurrent Neural Network for Network-wide Traffic Speed Prediction"*, In 6th International Workshop on Urban Computing, 2017.

[40] D. Shiebler,*"Music_RNN_RBM"*,https://github.com/dshieble/Music_RNN_RBM/blob/master/midi_manipulation.py, 2017, [Online]; Accessed: 05-07-2017

[41] N. Tran, T. Bui, N. Cheung, *"Dist-GAN: An Improved GAN using Distance Constraints"*, European Conference on Computer Vision, 2018

[42] M. Arjovsky, S. Chintala, L. Bottou, *"Wasserstein GAN"*, Proceedings of the 34th International Conference on Machine Learning, vol. 70, pp. 214-223, 2017

[43] M. Arjovsky and L. Bottou. *"Towards principled methods for training generative adversarial networks"*. In International Conference on Learning Representations, 2017.

[44] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung , A. Radford, X. Chen , *"Improved Techniques for Training GANs"*, Proceedings of the 30th International Conference on Neural Information Processing Systems, pp. 2234-2242, 2016

[45] T. Mikolov, K. Chen, G. Corrado, J. Dean , *"Efficient Estimation of Word Representations in Vector Space"*, Proceedings of the International Conference on Learning Representations , pp. 1-12, 2013

[46] L. Yang, S. Chuo, Y. Yang, *"MidiNet: A Convolutional Generative Adversarial Network for Symbolic-domain Music Generation"*, Proceedings of the 18th International Society on Music Information Retrieval Conference, 2017

[47] M. Bretan, G. Weinberg, and L. Heck. *"A unit selection methodology for music generation using deep neural networks"*, Proceedings of the International Conference on Computational Creativity, 2017

[48] K. Goel, R. Vohra, J.K. Sahoo, *"Polyphonic Music Generation by Modeling Temporal Dependencies Using a RNN-DBN"*, International Conference on Artificial Neural Networks, Lecture Notes in Computer Science, vol. 8681, pp. 217-224, 2014

[49] T. White, *"Sampling Generative Networks"*, https://arxiv.org/abs/1609.04468v3, 2016

[50] J. Kim, *"DeepJazz"*, Using Keras and Theano for deep learning driven jazz generation https://deepjazz.io/, 2017

[51] I. Goodfellow, Y. Bengio, A. Courville, *"Deep Learning"*, MIT press, 2016

[52] R. Zaripov, *"On the algorithmic description of the process of composing music"*, In USSR Academy of Sciences, vol. 132, no. 6, pp. 1283-1286, 1960

[53] F. Lerdahl, R. Jackendoff, *" A Generative Theory of Tonal Music"*, Cambridge, MA: MIT Press, 1983

[54] I. Xenakis, *" Formalized Music: Thought and Mathematics in Music"*, revised ed. Pendragon, 1992

[55] F. Colombo, W. Gerstner, *"BachProp:Learning to Compose Music in Multiple Styles"*, https://arxiv.org/abs/1802.05162,2018,[Online]; Accessed: 19-07-2018

[56] H. Dong, W. Hsiao, L. Yang, Y. Yang, *"MuseGAN: Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment"*, Proceedings of the the 30th international conference on Innovative Applications of Artificial Intelligence, pp. 34-41, 2018

[57] J. Chung, C. Gulcehre, K. Cho, Y. Bengio, *"Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling"*, Proceedings of the 27th international conference on Neural Information Processing Systems, Deep Learning and Representation Learning, 2014,

[58] E. Waite, *"Generating Long-Term Structure in Songs and Stories"*, Blog,https://magenta.tensorflow.org/2016/07/15/lookback-rnn-attention-rnn, 2018,[Online]; Accessed: 12-07-2018

[59] D. Abolafia, *"A Recurrent Neural Network Music Generation Tutorial"*, Blog,https://magenta.tensorflow.org/2016/06/10/recurrent-neural-network-generation-tutorial, 2016,[Online]; Accessed: 21-10-2018

[60] A. Roberts, J. Engel, C. Raffel, C. Hawthorne and D. Eck, *"A hierarchical latent vector model for learning long-term structure in music"*, 2018, Proceedings of the 35th

International Conference on Machine Learning, Machine Learning Research, vol. 80, pp. 4364–4373. Stockholmsmssan, Stockholm Sweden: PMLR

[61] P. Vincent, N. Lewandowski, and Y. Bengio, *"Modeling Temporal Dependencies in High-Dimensional Sequences:Application to Polyphonic Music Generation and Transcription"*, Proceedings of the 27th International Conference on Machine Learning, 2012

[62] A. Huang, and R. Wu, *"Deep learning for music"*,arXiv preprint,https://arxiv.org/abs/1606.04930, 2016, [Online]; Accessed: 10-10-2018

[63] J. Koum, B. Acton, *"WhatsApp"*, Social messaging Application,https://www.whatsapp.com/, 2009

[64] F. Chollet, *"Keras"*, Deep neural network library, http://keras.io/optimizers/, 2015

[65] J. Weel, *"RoboMozart:Generating music using LSTM networks trained per-tick on a MIDI collection with short music segments as input"*, Bachelors Dissertation, University of Amsterdam, Faculty of Science, Science Park 904, 1098 XH Amsterdam

[66] H. Yu, J. Wang, Z. Huang, Y. Yang, and W. Xu, *"Video Paragraph Captioning Using Hierarchical Recurrent Neural Networks"*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 4584-4593, 2016

[67] K. Lackner, *"Composing a melody with long-short term memory (LSTM) Recurrent Neural Networks"*, Bachelors Dissertation, Institute for Data Processing Technische Universität München, Munich, Germany, 2016

[68] J. Hui,*"GAN-Wasserstein GAN and WGAN-GP"*, https://medium.com\/@jonathan_hui/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490, 2018, [online]; Accessed: 03-10-2018

[69] J. Kiefer, J. Wolfowitz, *"Stochastic Estimation of the Maximum of a Regression Function"*, The Annals of Mathematical Statistics, vol. 23, no. 3, pp. 462-466, 1952

[70] H. Chu, R. Urtasun, S. Fidler, *"Song from pi: A musically plausible network for pop music generation"*, Proceedings of the 5th International Conference on Learning Representations, workshop paper, 2017

[71] W. Clarke, transcribed by D. Dolby, *"Speed of Plough"*, No 217, Page 107 William Clarke of Feltwell, https://maryhumphreys.co.uk/pdf/WilliamClarketunes.pdf, 2010, [online]; Accessed: 27-09-2018

[72] C. Raffel, *"Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching"*,https://colinraffel.com/projects/lmd/, 2018,[Online]; Accessed: 04-06-2018

[73] F. Wilcoxon,*"Individual Comparisons by Ranking Methods"*, Biometrics Bulletin, vol. 1, no.6, pp. 80-83, 1945