NTNU
Norwegian University of
Science and Technology

Generating Blues Solos with Variational Autoencoders

Eirik Katnosa Sandberg

Master of Science in Computer Science
Submission date: June 2018
Supervisor: Björn Gambäck, IDI

Norwegian University of Science and Technology
Department of Computer Science

Eirik Sandberg

Generating Blues Solos with Variational Autoencoders

Master's Thesis in Computer Science, Spring 2018

Data and Artificial Intelligence Group
Department of Computer Science
Faculty of Information Technology and Electrical Engineering
Norwegian University of Science and Technology

Abstract

This thesis describes the design and implementation of a variational autoencoder that generates blues solos. The architecture of the variational autoencoder is capable of capturing long-term dependencies in musical data, which is verified in experiments. A dataset of MIDI solos was manually extracted from a corpus of MIDI songs in the genre of blues and used to train a Long Short-Term Memory Recurrent Neural Network. Tools for extracting musical information from MIDI files into a format that can be used for training a network were designed, implemented, and verified. Results show that the network is able to generate solos that have significant variations from the training data. Some of the generated solos are capable of being mistaken for real solos, and a few even outperform real solos in certain aspects. However, in most cases limitations on the system that lead to losses in musical information, and a limited dataset, inhibits the system's ability to produce solos that are perceived as blues.

Sammendrag

Denne masteroppgaven beskriver design og implementasjon av en variational autonencoder som genererer blues soloer. Arkitekturen som er designet er i stand til å fange opp langsiktige avhengigheter i musikkdata, som blir verifisert i eksperimenter. Nettverket blir trent på et datasett bestående av MIDI soloer som er manuelt uthentet fra en samling MIDI sanger i sjangeren blues. Verktøy for å hente ut musikalsk informasjon fra MIDI filer og transformere informasjonen til format som kan bli brukt til å trene et nettverk er designet, implementert, og verifisert gjennom eksperimenter. Vi finner at nettverket er i stand til å generere soloer som varierer betydelig fra treningsdataen. Noen genererte soloer er også i stand til å bli forvekslet med ekte soloer, og overgår de reelle soloene i visse aspekter. Systemets begrensninger til å bruke all musikalsk informasjon i MIDI filer, og begrensninger i treningsdataen, begrenser systemets evne til å generere soloer som kan bli oppfattet som blues.

Preface

This thesis is submitted to the Norwegian University of Science and Technology (NTNU) as a part of the requirements for the degree of Master of Science in Computer Science. The work on the thesis has been performed at the Department of Computer Science, NTNU, Trondheim. The thesis was supervised by Professor Björn Gambäck, and co-supervised by Dr. Marinos Koutsomichalis.

I would especially like to thank my supervisor Björn Gambäck for letting me take on this project, and the effort and time he put into the whole process. I would also like to thank my co-supervisor Marinos Koutsomichalis for his great feedback and thoughts during this project.

I also want to thank Borgar Lie for his expertise in PyTorch during the implementation process, and Magnus Sahlgren for his qualitative assessment of the solos generated.

Lastly, I want to thank all the people that participated in the survey.

Eirik Sandberg
Trondheim, 27th June 2018

Contents

List of Figures

List of Tables

1. Introduction

Computational creativity is a topic that has been explored for a long period of time, and there are many examples where computers are used for creative tasks such as generating images, poems, lyrics or music. Over the years, a vast amount of papers have been presented that explore these areas using different methods and technologies as they have become available. The digitalization of music has over the years not only made it convenient to listen to music but also made the data available for analysis and other usages, such as algorithmic music generation. With the power of Artificial Intelligence, we can now use the data available to extract information to generate new versions of songs based on the old songs. As computational power available to us has increased, we are able to take advantage of the enormous amount of data available, and use deep learning to solve different problems.

In this thesis, a system that is capable of interpreting musical data is introduced, which takes advantage of deep learning to learn musical foundations and generate new solos. A variational autoencoder is implemented for generating blues solos from a corpus of 382 blues solos. The term 'blues' in this thesis will be used according to what freemidi.org considers as blues music, and all the songs they have put under the genre blues on their website.

1.1. Project goals and research questions

In this section the main goals and the research questions for this thesis are presented.

G1: Develop the necessary tools to extract musical information from MIDI and use it to train a network

Develop an encoder that reads MIDI files and transforms the relevant information to a format that represents musical data, and that can be interpreted and used by a neural network. This goal also includes formatting the data back to MIDI from the format that can be interpreted by the neural network.

G2: Develop a variational autoencoder that generates blues solos

Variational autoencoders have been used for generating music before, but either on a large corpus of millions of melodies or on a very small dataset. A goal of this thesis is to design, develop and implement a variational autoencoder that can be trained on a corpus of blues solos, and generate variations of its training data as well as variations of the same model using different samples from its latent distribution. The output should not be identical to the solos in the training set but share some basic musical foundation.

R1: Can the generated solos from the system be mistaken for a solo from a real song?

The input solos are solos in MIDI format from real songs. Can a generated solo from the system, with an added backing track, be mistaken for a solo that is used as input to the system by a group of people surveyed?

R2: What factors have an effect on the network's ability to generate solos that are more similar to its input?

Various factors affect the output of the network. Experiments on different properties can reveal its impact on the system's ability to imitate its training data.

1.2. Contributions

The main contributions from this Master's Thesis are:

1. A proposed architecture for a variational autoencoder for capturing long-term dependencies in musical data.
2. Design and implementation of a variational autoencoder that is capable of generating blues solos.
3. An extensive evaluation of how different parameters of the network affect the output.
4. A quantitative evaluation of how the solos sound to people.

The code for the system designed in this thesis is available at:
`https://github.com/eiriksandberg/NeuralMusic`

The generated solos for this project is available at:
`https://drive.google.com/drive/folders/1YnI23CIOpihE2ho7Ls7giQwUerNnkxRK?usp=sharing`

1.3. Thesis structure

The thesis is structured in the following way: Relevant background theory that will enhance the reader's understanding of the material in this thesis will be presented in chapter 2. Next, a chapter on related work which describes similar work in melody composition using computers, including different methods, algorithms and techniques is presented. The architecture of the system is presented in chapter 4, followed by a chapter where the experiments conducted and the results are described and presented. Next, the results are evaluated in chapter 6, before discussed in depth in chapter 7. Lastly, a chapter which concludes the thesis and presents future work and limitations for the system is presented.

2. Background Theory

In this chapter, the reader will be provided with the necessary background theory that will enhance the understanding of the material discussed later in this thesis. The first section gives a brief introduction to selected parts within music theory which will enhance the overall understanding of this thesis. Next, sections describing Artificial Neural Networks and Recurrent Neural Networks with focus on Long-Short-Term-Memory networks are discussed, followed by a section on Variational Autoencoders. The section that follows, explain details of Musical Instrument Digital Interface, followed by a short section on creativity. The chapter is concluded with similarity measurements.

2.1. Basic music theory

This section describes basic music theory and provides the reader with necessary information to get a better understanding of the rest of the thesis. The models found under this section are based on guitar as the primary instrument, although the theory also applies to other instruments as well.

Notes

Western music generally uses 12 distinct notes. Guitarists understand these notes with respect to their positioning on the fretboard. These twelve notes are *A, A#/Bb, B, C, C#/Db, D, D#/Eb, E, F, F#/Gb, G, G#/Ab*. A note denotes the pitch and the duration of a sound, where the pitch allows ordering on a frequency-related scale. These notes can be placed out on the fretboard over the different strings as shown in figure 2.1. Notes have different versions of themselves which are called octaves. An octave of a note is a note that is played at half or double the frequency of itself. A guitar usually has 22 or 24 frets, where the notes repeat themselves only an octave higher than shown in figure 2.1 from the 12th fret. For this reason, we will only be showing figures of the fretboard up to the 12th fret.

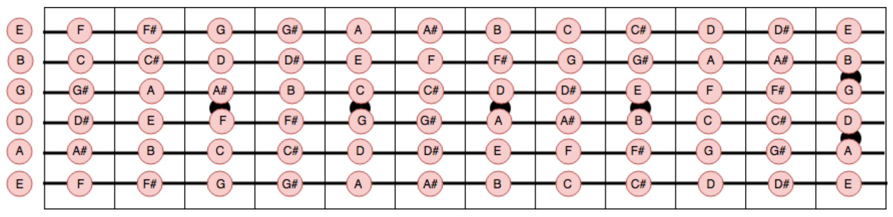

E F F# G G# A A# B C C# D D# E
B C C# D D# E F F# G G# A A# B
G G# A A# B C C# D D# E F F# G
D D# E F F# G G# A A# B C C# D
A A# B C C# D D# E F F# G G# A
E F F# G G# A A# B C C# D D# E

Figure 2.1.: All tones found on the guitar fret

Musical key

Khan Academy defines a musical key as *"A group of pitches based on a particular tonic, and comprising a scale, regarded as forming the tonal basis of a piece or section of music"* [13]. An example of this is the first position in the minor pentatonic scale played in the key of A. The same position applies also to different keys, but the root note will change depending on the key in which the scale is played.

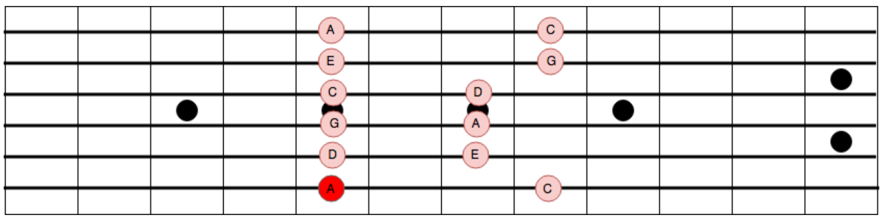


Figure 2.2.: Pentatonic scale in the key of A

Tempo

Khan Academy defines tempo as *"the rate of speed of a musical work"* [13]. Tempo is measured in beats per minute (bpm). If a song has one beat every second, its tempo is 60 bpm.

Duration

A note or pause has a duration which is represented as a fraction. Normal values for durations are $\frac{1}{1}, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}$. A $\frac{1}{1}$ note is called a whole note and $\frac{1}{2}$, a half note, where its duration is half the length of a whole note. $\frac{1}{4}$ is called a quarter note, and its duration is a quarter of a whole note, and so on.

Time Signature

Time is divided into various measurement's e.g., hours, minutes and seconds. When talking about music, however, we divide it into beats. The beat can be thought of as the pulse of the music. The time signature is represented as a fraction, where the top number states the number of beats to count, while the bottom number indicates what kind of note to count, i.e., if the beats should be counted as quarter notes, 8th notes, 16th notes, and so on. A $\frac{4}{4}$ time signature is found in most songs. Waltz has a well known time signature of $\frac{3}{4}$, while it is not uncommon to find jazz with time signatures $\frac{5}{4}$ or $\frac{7}{4}$ [32].

Microtones

When talking about blues, and specifically guitar, microtones play a huge part. When listening to artists such as B.B. King, Eric Clapton and John Mayer, microtones are heard all the time. When playing the guitar, a very common technique is to bend notes, which means to bend the string on the fretboard to gradually increase the frequency of the string, and hence change the pitch of the note. By bending the string, it is possible to increase the pitch in a continuous interval, which means that it is possible to create notes that are not discrete like an A or C#. These are called microtones.

Consonance and dissonance

Consonance is a measure of how pleasant an interval or a chord sounds, while dissonance is the opposite.

Scales

A scale in music is a set of notes that are ordered by pitch. There exists a vast number of scales, where some of the more common are the major scale, the major pentatonic scale and the minor pentatonic scale.

The minor pentatonic scale

A very common scale found in most songs, but especially in blues, is the minor pentatonic scale. The minor pentatonic scale contains 5 notes, hence *pentatonic*, and can be found in figure 2.3 in the key of E. There are three colors in the figure that represent different aspects of the scale. The red mark the key of the song which is E, the bright red are notes that are in the scale, while the blue notes are 'blues' notes that are a part of the *blues* minor pentatonic scale. The blues scale is very popular in blues music and contains an additional note to the scale that is commonly heard in blues music, and in blues solos.

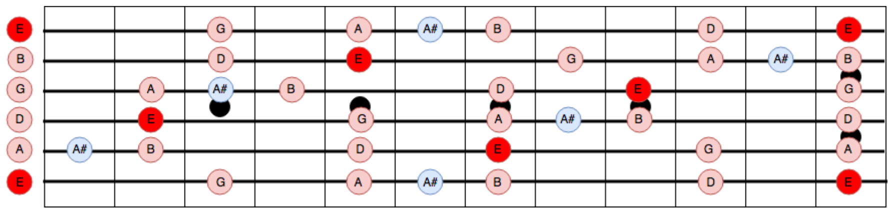


Figure 2.3.: Blues minor pentatonic scale in the key of E

2.2. Artificial Neural Networks

Artificial neural networks (ANN) are networks that take architectural inspiration from the brain. The network's behavior is to "learn" a particular task or to improve with time when trained with examples. ANNs can be looked at as functions. The networks take an input, process it, and produce an output based on the input. These networks can be used for tasks such as image recognition, classification problems or problems linked to computational creativity, among a vast number of other usages. There are many different types of neural networks, but for this project, we will focus on recurrent neural networks that are described in section 2.3.

2.3. Recurrent Neural Networks

Recurrent neural networks (RNN) are networks that process sequential data. Normal feed forward networks work well for classifying problems, but do not have the feature of remembering previous states. RNNs, however, are capable of handling this problem [30]. Figure 2.4 shows how an RNN works by taking an input x_t and outputing a value h_t. The information from the previous step is passed on to the next step. The repeating module usually is a single layer with a simple structure, such as a *tanh* layer. The *tanh* activation function, is used to determine the output of the neural network within values between -1 to 1 and is given by equation 2.1.

$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1 \tag{2.1}$$

However, traditional recurrent neural networks are not capable of long-term memory, and remembering information that was fed into the network a while ago [31]. Fortunately, this problem is something networks handle. [14, p. 367-415]

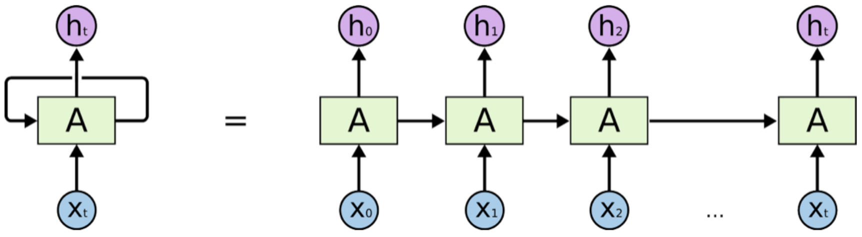


Figure 2.4.: Recurrent Neural Network. Taken from http://colah.github.io/posts/2015-08-Understanding-LSTMs/ with permission from Cristopher Olah

Long Short-Term Memory (LSTM) are recurrent neural networks that remember information for a long period of time. Similar to recurrent neural networks, LSTM networks have a chain-like structure as seen in figure 2.4, but the repeating module is different in an LSTM network compared to a regular RNN, which usually uses a single layer with a *tanh* function or something similar. LSTM networks have four layers in the repeating module, and the structure of a single LSTM cell can be found in figure 2.5 [31].

The top line in figure 2.5 is the cell state, and information can either be added to or removed from the cell state through the gates in the cell. The first gate is called the forget gate, which decides how much from the previous cell state that should be remembered. A *sigmoid* function is applied to the gate, which outputs a value between 0 and 1. The *sigmoid* activation function can be found in equation 2.2.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

If the value is zero, it means that everything should be forgotten, while if the value is one, it means that everything should be remembered from the previous state. Next, there are two parts. The first part is a *sigmoid* layer which is called the input gate layer. This layer determines what values that should be updated, while the second layer, a *tanh* layer, creates candidate values that could potentially be added to the state. The next step is to actually perform everything discussed previously in this section: Forget what needs to be forgotten and update the values with the candidate vectors multiplied with how much update that is desired. The final step is to run a *sigmoid* layer, which will decide what parts of the cell state to output. The cell state is put through a *tanh* function and multiplied with the output from the *sigmoid* function to only output the wanted parts. [31]. The equations that are computed are found below.

$$\begin{aligned}
f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) && \text{Forget gate}\\
i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) && \text{Input gate}\\
g_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) && \text{Candidate values}\\
C_t &= f_t \cdot C_{t-1} + i_t \cdot g_t && \text{Cell state}\\
o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) && \text{Output gate}\\
h_t &= o_t \cdot \tanh(C_t) && \text{Hidden state}
\end{aligned}$$

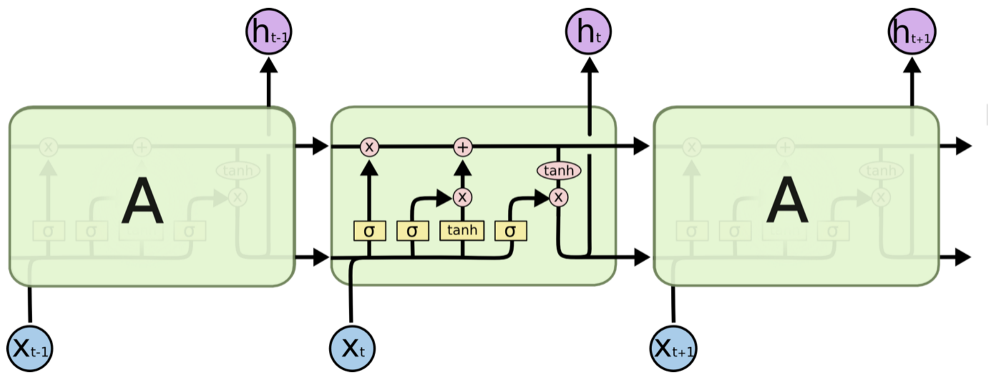


Figure 2.5.: LSTM. Taken from http://colah.github.io/posts/2015-08-Understanding-LSTMs/ with permission from Cristopher Olah

2.4. Variational Autoencoders

Variational Autoencoders (VAE) are models that use Bayesian inference for generating data. The underlying probability distribution is modelled, sampled, and used for generating new data.

A probabilistic model of this type has latent variables and/or parameters that have intractable posterior distributions. Kingma et al. [20] proposed a solution called the Variational Bayesian (VB) approach, which uses a reparameterization trick on the variational lower bound, which yields a differentiable unbiased estimator on the lower bound. This estimator can be optimized using standard stochastic gradient descent techniques.

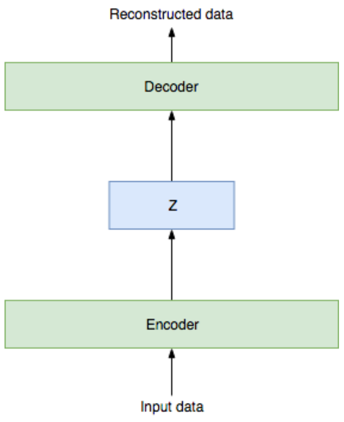


Figure 2.6.: VAE parts

A variational autoencoder consists of three parts, an encoder network, a Gaussian distribution z, and the decoder network, as seen in figure 2.6. The encoder network is a neural network which outputs the mean covariance, $\mu_{z|x}$, and the diagonal covariance, $\sum_{x|z}$. These values are used for sampling from the latent distribution, z. Latent information from the training data needs to be captured in order to describe features that can be extracted from the input to the variational autoencoder. The latent distribution needs to capture dependencies in the input data in order to be able to generate similar data as the input. A VAE can generate data by sampling from $\mathcal{N}(0, I)$, where I is the identity matrix [7].

If a VAE is trained on a set of images of human faces, things like the mouth, eyes and nose will be typical features. By varying sampling values from the latent distribution, the amount of a smile or the placement of the eyebrows can, for instance, be controlled in the generated image. After sampling from z, one or more networks are used to transform the data back to the original input dimension. This is done through the decoder network [10].

To optimize the estimator, standard stochastic gradient descent techniques are performed on the tractable terms. Equation 2.3 shows how to derive the differentiable terms for the lower-bound estimator.

$$\begin{aligned}
\log p_\theta(x^{(i)}) &= \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \qquad (2.3)\\
&= \mathbf{E}_z\left[\log\frac{p_\phi(x^{(i)} \mid z)p_\phi(z)}{p_\phi(z \mid x^{(i)})}\right]\\
&= \mathbf{E}_z\left[\log\frac{p_\phi(x^{(i)} \mid z)p_\phi(z)}{p_\phi(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right]\\
&= \mathbf{E}_z\left[\log p_\phi(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log\frac{q_\phi(z \mid x^{(i)})}{p_\phi(z)}\right] + \mathbf{E}_z\left[\log\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right]\\
&= \underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \parallel p_\theta(z))} + D_{KL}(q_\phi(z \mid x^{(i)}) \parallel p_\theta(z \mid x^{(i)}))
\end{aligned}$$

Only the first two terms that are highlighted above are used for the estimator because the last term is intractable. However, since the last term is a KL divergence [17], we know that the term must be ≥ 0, and therefore we have a lower bound and can find the values for θ and ϕ that maximize the lower bound [10].

After training, the encoder is simply removed from the VAE to generate new samples of data. This is done by sampling values of z in $\mathcal{N}(0, I)$ and input these into the decoder. By varying values for z, samples can be generated based on different features [7].

2.5. Musical Instrument Digital Interface (MIDI)

In the music industry in 1981, talk of a 'universal' digital communication system first began. Competitors in the synthesizer industry started to worry that a lack of compatibility between manufacturers would restrict the usage of synthesizers. The same year a proposal of such was made, which was later further improved under a collaboration between an American and a Japanese manufacturer. In 1982, the term MIDI (Musical Instrument Digital Interface) first appeared in public, which is also the year it was first used on an instrument [26]. MIDI is a leading industry music protocol for musical data from MIDI controllers to computers and outputs music. MIDI is used every day by various artists, professionals and amateurs on computers, smartphones and tablets all around the world. MIDI controllers are controllers that support the MIDI protocol and have some sort of interface that makes it possible to interact with the controller to produce music. An example of this is a MIDI keyboard or a drum machine [25].

2.5.1. MIDI general

The MIDI protocol specifies that there are 128 possible pitches that can be played, which means that a MIDI song can contain at most 128 unique pitches. A MIDI song consists of one or more tracks that together make up a song. It is normal that one track is a single instrument, but it is possible to change the instrument of a track at any given time in the track. The timings in MIDI files are all centered around ticks and beats, and the time attribute in MIDI messages is measured in ticks, which is the smallest unit of time found in MIDI. Every MIDI song has a value called ticks per beat which is a factor that determines how many ticks that are associated with a duration of a note. Different MIDI songs can have different ticks per beat values. A note has a duration, but the duration in ticks is related to the ticks per beat value. This means that a 16th note in one song can have a different duration in ticks compared to a 16th note in a different song. The ticks per beat standard value is 480.

2.5.2. MIDI messages

A MIDI file contains several tracks which each represents an instrument. Each track has a series of MIDI-messages that contains information like tempo, type of instrument, what pitches are being played, and so on. For this project, there are two types of messages that are particularly relevant, the *note_on* and *note_off* messages. The message type *note_on* describes what pitch should be played and its velocity. The message *note_off* follows a *note_on* message to indicate that the pitch should not be played anymore. *Time* is a very important component of MIDI messages. All MIDI messages have a time value linked to them that indicates the time until the next message should be processed, the unit of which is ticks.

There are also other messages in the MIDI protocol, two of which that are relevant for this thesis, *program_change* and *pitchwheel*. *program_change* contains information about what instrument the current track is. The instrument is identified with an integer that maps to a specific instrument. The sound of the instrument varies based on the program the MIDI file is opened with. *pitchwheel* is a message that follows a *note_-on* message. It contains information on how much higher or lower the frequency of the pitch in the *note_on* message should be altered. This message makes it possible to create musical pieces with pitch bending, which in guitar terminology is called string bending, and is further described in section 2.1. This type of message hardly occurs alone and is often part of a group of consecutive MIDI messages. Pitch bending in practice is often a continuous motion for a given duration, which requires a series of MIDI messages to simulate bending of a guitar string over a period of time.

2.5.3. MIDI repositories

MIDI is often used in systems that generate chord progressions, melodies or percussion, etc. It is a structured way of representing musical data. Different systems are trained on different types of music, and there exist repositories based on genres, artists, TV-shows and video games to mention a few. Given its popularity, there are a vast amount of repositories online, where one can download MIDI files in different genres and by different artists. Examples of such are freemidi.org and midiworld.com. In this thesis, the repository of freemidi.org is used, and all songs from the blues section were downloaded.

2.6. Creativity

There are many definitions of creativity, and perceptions of what creativity is. Howard Gardner defines creativity as *"the ability to solve problems, or to fashion products, that are valued in one or more cultural or community settings"* [12]. Bill Lucas stated in [21] that *"Creativity itself is notoriously difficult to define"*, but still defined creativity as *"a state of mind in which all of our intelligences are working together. It involves seeing, thinking and innovating"*. In this thesis, Ken Robinson's definition of creativity is used. He defines creativity as

> *"Imaginative processes with outcomes that are original and of value"* [34].

In the book Unlocking Creativity: Teaching Across the Curriculum [11], three processes of creative evolutions are presented:

- Generation
- Variation
- Originality

Generation is to create, form or bring something into being. However, it is not sufficient to create something and call it creativity, which leads us to the second process, *variation.* The created work need to be somehow different than existing ones. It can not simply be reproduction or a copy of existing work in order to be creative. Variation is an exploratory process where the output adds to or is a variation of that which is given, sometimes with success, others with failure. *Originality* can be looked upon as the effect of surprise. The generated work with applied variation can result in something that some perceive as "something that I have never heard or experienced". Originality comes in different degrees, which are *individual, social* or *universal.* Individual is originality to a person's previous thought, social is originality to a group's previous thoughts, and universal is originality in terms of all previous human experience [11].

2.7. Similarity measurements

Similarity between the generated data and the input data can be analyzed to see how creative a system is, and a measure of similarity can help describe variations and originality. In [29], Müllensiefen and Frieler have experts listen to a number of different songs with small variations, and rank the songs based on how similar they are to one another. Next, they use different algorithmic approaches to measure the similarity between songs and compare the results with the results from the experts. They find that for pitch and interval measures, edit distance and N-gram give the best results compared to the experts.

2.7.1. N-gram comparison

N-gram comparison is based on splitting a query sentence into overlapping substrings of N characters or words in length, and then comparing it to some other string. The N-gram similarity is given by equation 2.4.

$$Sim = \frac{\sum samegram}{\sum allgram} \tag{2.4}$$

If we have a query string, we can split that sentence into N-grams, and compare it to a string which we also split in N-grams. The items that share at least one N-gram are collected, and the items are ranked by the score based on the ratio of shared to unshared N-grams between strings [1].

2.7.2. Edit distance

Edit distance or Levenshtein distance is a metric for measuring the difference between two strings. It is the minimum number of single character edits required to change one string into the other. There are three legal edit operations:

- Insertion
- Deletion
- Substitution

Insertion allows you to insert a character, *Deletion* allows you to delete a character, while *Substitution* allows you to exchange one character for another.

3. Related Work

Generating music with algorithms has been a popular topic in the field of computer science for a long time. This chapter will provide the reader with an overview of the more popular applications for computational creativity in terms of musical melodies. First, a section on melody composition using grammar is presented, followed by approaches using Markov chains. Next, a section describing how evolutionary algorithms are used in music generation is presented before a section on the same application using deep neural networks. The neural network section is subdivided into two parts, one describing related work with the use of feed forward networks and recurrent neural networks, and the other describing related work linked to variational autoencoders.

3.1. Music composition using grammars

Terry Winograd applied techniques developed by Noam Chomsky from the field of linguistics to music as early as 1968 [38]. Winograd explains that linguistics and music share some similarities which allow us to express music with the same techniques as we have previously expressed sentences in natural language. Natural language have grammars that determine what words that can succeed the previous. Music, however, follows a set of rules which determines what pitches that can succeed the previous. The foundation of music theory can act as grammar, which will determine what “legal” pitch that should succeed the previous. This is the foundation of music composition using grammars. Jon McCormack [23] adapted string rewriting grammars based on L-Systems, which was previously mainly used in biological systems and morphogenesis, and adapted this to music composition. An L-system is a form of string rewriting grammar that is used to provide a compact way to represent complex patterns. Keller et al. [18] proposed a grammatical approach to music generation. They showed that by using probabilistic grammars, they were able to assist a soloist through a software with the creation of jazz solos over a chord progression. Probabilistic grammar adds probability to the grammar which allows some control over the sequences chosen by the system. Higher probability results in more “mainstream” sequences, while lower probabilities results in less common sequences. In addition, they showed promising results in generating an improvised jazz solo real time using grammar. The musical compositions using grammars will give correct predictions, and produce melodies and chord progressions which sound pleasing, due to the rule-based implementation.

3.2. Markov chains

Markov chains have been widely used in music composition due to the sequential dependencies music has, and were very popular at the beginning of algorithmic composition. Moorer [27] used Markov chains already back in 1972 to generate short melody fragments that captured some melodic characteristics, but they suffered from what Moorer described as 'sounding Alien-like'. More recently, Bell [2] created a system using Markov chains in combination with evolutionary algorithms to choose next pitch, rhythm and chord. He used Markov chains in order to choose the next pitch, which is a method for determining the next state of a sequence solely based on its current state, and knowing the whole history of the sequence. He used a genetic algorithm to find the set of Markov chains that sounded more pleasing to the ear. Although the system did a good job at composition, it lacked the organic nature of human-composed music. Music composition using Markov chains requires a deeper understanding of music theory that cannot always be taken for granted and that may prove an important limitation of the system in certain contexts.

3.3. Evolutionary algorithms

John. A. Biles is a name that is often mentioned when talking about computational creativity and music generation using computers. Biles created GenJam [3] which is a genetic algorithm-based model of a novice jazz-musician learning to improvise. GenJam uses mappings from chords to scale to generate jazz solos. These are mappings that are based on jazz music theory that map chords to a scale, and the notes that are found in that scale. A human actor is responsible for fitting the system by typing one or more 'g' or 'b' depending on how good or bad the generated solo is perceived. Biles also extended the system in 1998, enabling GenJam to listen to a real person playing, and to respond to what was played [4]. More specifically, the system and the person played four measures each, and while the human was playing, GenJam listened to what was played, and based its next generated solo on what it had just heard. Biles' work was one of the motivations for this project, and his ability to generate trumpet solos in the style of jazz is highly relevant for my thesis. However, where Biles generate solos for trumpet in the genre of jazz, the present work aims to generate solos based around guitars in the genre of blues.

Marques et al. [22] developed a system that took advantage of genetic algorithms to search for a solution in a large search space. The system had a fully automated fitness function, which means that there was no interaction between actors to determine what sounded good or bad. The fitness function evaluated harmony, tone and melody. Marques et al. concluded that they were able to generate short sequences of music that sounded pleasing to the ear of the listeners during the experiments they conducted. The paper by Marques et al. is relevant because of its independence from human interaction to produce pleasing melodies. However, they claim that the generated melodies sound pleasing to the listeners during the experiments, but fails to describe what they mean by pleasing, who, and how many the listeners are.

3.4. Music composition using deep neural networks

The first neural networks were described already in 1943 by Warren McCulloch and Walter Pitts [24]. However, the first successful implementation on a computer did not take place until 1959 with the introduction of *ADALINE* and *MADALINE* [37]. Since then, there have been major improvements in the field of neural networks and deep learning. Music generation is a sequential problem where the next note is dependent on the previous. The introduction of Recurrent Neural Networks, RNNs, described in section 2.3, have the ability to remember previous states and were first introduced in 1985 by Rumelhart et al. [35].

3.4.1. Feed forward networks and recurrent neural networks

Michael Mozer implemented a recurrent autopredictive neural network which extracts stylistic regularities from its training data in 1994 [28]. This network was trained on Bach pieces and traditional European folk melodies, and was able to compose new pieces. A problem with the system, named CONCERT, was that it failed to capture a global coherence. CONCERT is a recurrent network that once trained and fed an initial start pitch can generate pieces of its own, by predicting the next pitch based on the previous. Several methods were explored to overcome the limitation of the lack of global structure, but Mozer concluded that note-by-note prediction for music composition was doubtful.

Douglas Eck and Jürgen Schmidhuber [8] generated 12-bar blues pieces using an LSTM network. The system can either generate melodies in a fixed note interval or chord progressions based on training data from a corpus of 12-bar blues songs. The melody generation presupposes that the chord progression does not change and remains the same. The generated melody is described to have a “bluesy feeling”. This is the only paper to my knowledge that tries to generate blues music using LSTM networks, which is what this thesis is about.

Hadjeres and Briot [5] implemented a simple feed forward network that generates melodies based on melodies learned from its input. The system was called MiniBach and was later expanded into DeepBach [15], a more advanced architecture which is specialized for Bach chorales. DeepBach combines two LSTM networks and two feed forward networks to generate music. The architecture can also write music backwards, taking notes from “the future” into account and composing the future note’s predecessor, which also happens to be a technique used by Bach while composing music. Papers that describe melody generation using neural networks, usually take advantage of RNN networks and more recent papers almost exclusively use LSTM networks due to their ability to remember previous states and capture long-term dependencies. This is a critical factor when designing such systems, and is highly relevant for this thesis.

Brunner et al. [6] created a music theory aware chord based generator using LSTM networks. The architecture consists of two LSTM networks where the first network predicts a chord progression which is later fed into a second network to generate polyphonic music. The system is able to capture long-term structure with the use of LSTM networks, and is also capturing musical theory in terms of chords that relate. The work of Brunner et al. shows that it is possible to extract musical foundations out of training data, which is very relevant to this thesis.

3.4.2. Variational Auto Encoders

In 2014 Kingma and Welling [20] showed how efficient learning can be achieved in probabilistic models with continuous latent variables and intractable posterior distributions. This made it possible to use stochastic gradient descent techniques to estimate the probabilistic distribution. Variational autoencoders have increasingly been used for generative problems after this paper was published. Although a majority of problems that take advantage of the generative properties of variational autoencoders are image related, there is also some use of it in music.

Fabius et al. [9] implemented a variational autoencoder with stochastic gradient descent techniques on music from video games represented through MIDI files. They implemented what they called a Variational Recurrent Autoencoder (VRAE), which is an RNN model based on a Variational Autoencoder. The VRAE allows mapping of time sequences to a latent representation which enables efficient, large-scale unsupervised variational learning of time sequences. For their experiments, they used 8 MIDI files from well-known video game songs. They showed that it was possible to train their VRAE with Stochastic Gradient Variational Bayes and how the different songs were represented in the latent space. The use of VAEs in music generation is highly relevant for this thesis. The paper by Fabius et al. gives a deeper insight into how different pieces of music are represented in the latent dimension, but does not provide detailed information on the implementation of the latent code.

Tikhonov et al. [36] created a similar system, but represented the data in a different way. They used one-hot encoding to encode pitch, octave and duration. They also used techniques from natural language processing by encoding the data in the same way as text strings. Word-based generation of text strings results in an extensive multidimensionality due to all the possible combinations of words that relate semantically and grammatically. Instead, Tikhonov et al. used character-based generation, which generates letter-by-letter rather than word-by-word, which significantly reduces the multidimensionality of the state space. They used a 4-layer LSTM network for both encoder and decoder. The system predicted the n_{k+1} note based on $n_1, ..., n_k$ previous input. They failed to assess the quality of the output due to the individual perception of what high-quality music is. However, they described an architecture which allows control over the style of output due to the latent distribution.

Hennig et al. [16] used a variational autoencoder to generate polyphonic music. In addition, they used an LSTM network to capture the long-term dependencies to create music. Hennig et al. introduced a variation to their VAE to control the key of the song that they were generating, which they called a classifying VAE. The classifying VAE included a classifier which was trained concurrently with the generative model to infer the class of each data point. VAEs are not capable of having discrete latent variables which makes it hard to infer the class (key) of the generated song, and by introducing the classifier to the VAE, they were able to infer the discrete class by introducing a continuous latent variable.

Roberts et al. [33] developed a recurrent variational autoencoder to reproduce short musical sequences. Their system was developed with Tensorflow and they initially focused on developing a model for 2-bar tracks used for generating drum tracks and melodies mainly used as backing tracks. The melody loops were represented as a sequence of 32 categorical variables taking one of 130 discrete states. 128 of these states represent one of 128 note-on pitches while the last two are rest and hold states. The drums are represented as 32 categorical variables taking one of 512 discrete states, where the states are combinations of all possible drum sounds, similar to word-based generation discussed earlier in this section. After they successfully reconstructed the 2-bar pieces, they switched focus to reproduce 32-bar lead melodies. The architecture of the variational autoencoder has an encoder, a latent distribution and two layers of decoders. The encoder consists of a single-layered bi-directional LSTM network, where they use half the output as μ and the other as σ which is used to parameterize a 512 dimension multivariate Gaussian distribution with a diagonal covariance matrix used for z. The decoder takes a z, which is passed through a linear layer the initialize a 2-layer LSTM which outputs 16 embeddings. These embeddings are passed through another linear layer to initialize another LSTM layer that produces individual 16th note events. Each embedding is used as an input to the lower level LSTM network which each produces individual 16th note events for 2 bars each. The 16 embeddings that each produce two bars, are concatenated at the end to produce a 32 bar melody. Lastly, the 32-bar melody generation are combined to generate bass- and lead melody tracks with the 2-bar loop architecture to produce trios. The paper by Roberts et al. is the most relevant paper for this thesis, and it has inspired a great deal of the implementation in the system presented. Their system is implemented using Tensorflow, while the system presented in this thesis is implemented using PyTorch. Roberts et al. does not go into implementation details of their system but provides an overview of the system's architecture. The variational autoencoder they have implemented does generate 32-bar melodies, which is very relevant and requires an architecture that is capable of capturing long-term dependencies. The blues solos for the system described in this thesis do not need to be 32-bars, thus lighter requirements to the architecture's ability to capture long-term dependencies apply. This results in different implementations of the decoders.

4. Architecture

This chapter describes the architecture of the variational autoencoder implemented, as well as other components used for generating solos. The MIDI encoder is described first, followed by a description of the MIDI decoder. Next, a section on the training data is provided before the architecture of the variational autoencoder is explained along with each of its components.

4.1. MIDI encoder

When implementing the MIDI encoder, several objectives need to be achieved in order to successfully have the neural network interpret music from the MIDI file:

O1. A neural network needs to interpret the format of the encoded MIDI file.

O2. A solo can have chords, which are multiple pitches being played at the same time.

O3. Pitches can be held for a longer period of time.

O4. Training data needs to be quantized to account for imprecisions from the artists.

4.1.1. Format of the neural network input

The MIDI-protocol specifies that there are 128 possible pitches that can be played. The architecture of the MIDI encoder incorporates 128 fields, one for each pitch, and two special fields in addition to this. This way of representing the MIDI files for training a neural network is taken from Roberts et al. [33]. The two additional fields represent special cases where the first one is indicating a pause state, while the other is indicating a hold state. If the pause field's value is set to 1, it means that no pitches should be played or held at that timestep. If the hold state is set to 1, it means that one or more pitches should be held for an additional timestep. In addition to the hold state to be set to 1, the pitches that should be held also need to be set to 1. A solo track is divided into 16th notes, which means that one timestep is equal to one 16th note in the track. For each timestep in the solo, i.e., one 16th note, the row number that equals the pitch number is set to 1 in the encoded matrix. An example of this is if pitch 50 is played, the value at row number 49 is set to 1 (index in matrix starts at 0). All other rows are set to 0. The result will be a matrix with 130 rows, where one of the row's value is set to 1, while the rest of the row's values are set to 0. If a chord is played, all the pitches that are present in that chord are set to have value 1 for that timestep. The format of the encoded MIDI-file is a $130 \times n$ matrix where the 128 pitches and 2 additional fields are rows indicating one timestep, and n is the length of the solo in 16th notes. By introducing the format discussed above, objective **O1, O2** and **O3** found in the list in section 4.1 are handled. The MIDI encoder was implemented in Python using MIDO, a framework for working with MIDI files in Python. NumPy and Pandas were also used as well as several other libraries and packages. The format of the encoding is specified in the matrix below, where each column represents a timestep.

$$\begin{bmatrix} P_{1,1} & P_{2,1} & \dots & P_{n,1} \\ P_{1,2} & P_{2,2} & \dots & P_{n,2} \\ P_{1,3} & P_{2,3} & \dots & P_{n,3} \\ \vdots & \vdots & \vdots & \vdots \\ P_{1,128} & P_{2,128} & \dots & P_{n,128} \\ Pause & Pause & Pause & Pause \\ Hold & Hold & Hold & Hold \end{bmatrix}$$

4.1.2. Quantization

Before the blues solos can be encoded to the format discussed in section 4.1.1, imprecise timings from the artists need to be removed with quantization. The reason for this is that imprecise timings need to be eliminated in order for the preprocessing to properly divide the solos into 16th notes. Most of the solos that are used for training are recorded by people and are therefore not completely accurate in terms of the timing of hitting a note. Figure 4.1 illustrates how the timing of notes are corrected before preprocessed and divided into 16th notes.

Every solo has a *ticks per beat value* which will impact the MIDI message times. Some MIDI solos have smaller values for the MIDI message times, that need to be handled in order to quantize the solos. This is directly linked to the ticks per beat value. The standard time interval between two 16th notes expressed in MIDI time is 120. This means that one 16th note's duration is 120 when the ticks per beat value is set to standard. However, this varies from solo to solo and needs to be taken into account before the tracks are quantized. Equation 4.1 below calculates the MIDI time for a $\frac{1}{16}$ note.

$$factor = \frac{120 \times tpb_{solo}}{tpb_{standard}} \tag{4.1}$$

Now, only a simple modulo operation is needed to find out if the note needs to be quantized. This is done with the following equation, 4.2

$$rest = t_{MIDImessage} \mod factor \tag{4.2}$$

This modulo operation discovers if the note is struck too early or too late, or held for too long or too short. Based on the result, the note is corrected to perfect 16th notes, by cutting or adding the *rest time* necessary to achieve perfect timings. This step handles objective **O4** listed in section 4.1.

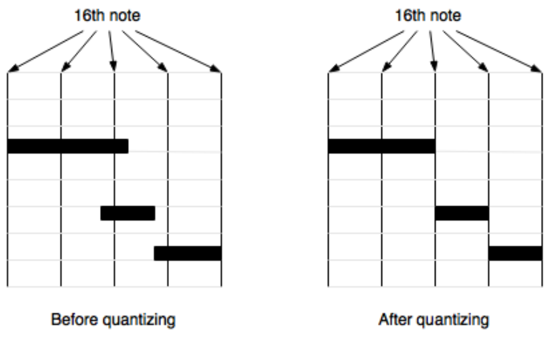


Figure 4.1.: Before and after quantizing

The training data has been cleaned manually to ensure high quality of the data. This choice eliminated a track selection step that was previously implemented in the system. The track selection algorithm analyzed each MIDI-track in a MIDI file and determined if the track was going to be used for training. However, when the cleaning of the dataset was done manually, the need for the track selection algorithm disappeared, because the action is performed simultaneously as the dataset is cleaned. After the solo has been encoded into the proper format, it gets written to a text file and saved. These text files are later read by the variational autoencoder as input.

4.2. MIDI Decoder

The data generated from the neural network is not directly readable and needs to be decoded back to a MIDI file so it can be inspected and listened to. When recreating the MIDI file, it only recreates the *note_on* and *note_off* messages because information like pitch bend, change in tempo, or program change was intentionally left out of the implementation. The output from the VAE is a $n \times 130$ matrix, where n equals a timestep, which represents a $\frac{1}{16}$ note. For each n in the matrix, the following steps are performed:

1. Check if state is hold or pause state. If so, extend the MIDI time messages for the relevant pitch. This is either a *note_on* or *note_off* message.

2. If the note is not a hold or pause note, add the note to a list that keeps track of notes that are not yet ended. The list contains an overview over notes whose duration is longer than the current timestep. This is a necessary implementation detail, which makes it possible to append *note_off* messages in future timesteps.

3. End notes that are finished. If a note has ended, we need to append a *note_off* message to the MIDI message list.

The instrument is by default set to program 27, which is a jazz guitar, but this has no impact on the actual output other than the instrument that will be set as default when opening the file in a program that can read MIDI files. After the entire output produced by the neural network is processed, and MIDI-messages have been generated, the framework, MIDO is used to generate the MIDI file. The framework takes in a list of MIDI messages and generates a MIDI file based on the messages the list contains. The result is a complete MIDI file that can be listened to when opened in a program that reads MIDI files. The same frameworks, libraries and packages as used in the MIDI encoder are also used for the MIDI decoder.

4.3. Training data

The training data is a dataset from freemidi.org. The songs selected are all songs found under the 'blues' section on the website. It contains artists such as Chuck Berry, Alicia Keys, Adele, Bob Dylan, B. B. King, Bee Gees, Eric Clapton, George Ezra, Jimi Hendrix, Stevie Ray Vaughan, Rod Stewart, Tom Jones, Gary Moore and Tina Turner, to mention a few. There are many different artists from different decades that play different styles of music in this dataset. The music spans over decades, where artists such as Muddy Waters, Chuck Berry and B. B. King started their active careers in the 40s, to artists such as Adele and George Ezra that released their first albums in 2008 and 2014, respectively. The dataset is a collection of songs that are not necessarily considered blues by all people, and thus, it are very varied.

import.io was used to generate download URLs for all songs found under the blues section. The songs in a respective genre shares parts of the same URL, which made it possible to create a regular expression to extract the parts of the URL which are dynamic. import.io provided tools to generate the download URLs to all of the songs, which were later downloaded in batches. Efforts to extract the guitar and bass tracks only from the MIDI files were made, but the selected tracks contained much noise in the form of being non-:solo tracks, as well as excluding most guitar solos because the guitar solo tracks were encoded in a different instrument than a guitar instrument in MIDI. Detecting what could be a solo in a MIDI song is also a very hard task because there are no rules to what a solo is. A MIDI song contains many different tracks with different instruments that are not always the correct instrument played in the actual song, e.g. a guitar track can be encoded as a synth instrument. In addition, it is very hard to detect a guitar solo with an algorithm, because many none-solo tracks share the same similarities as solo tracks. Efforts were made to create a rule-based system that could find the tracks which contained solos, with rules such as:

- Eliminate tracks with MIDI instruments that are not guitar or synth tracks
- Compare number of pitches played
- Compare number of pitches played over a given time to detect faster playing and more frequent pitch changes.

After spending some time trying to create such an algorithm, the work was stopped due to unsatisfactory results and an evaluation of time needed to create such a complicated algorithm. The final decision was to clean the dataset manually, ensuring high-quality training data. Each MIDI song was opened in Logic Pro and manually checked for guitar solos. If a guitar solo was found, the solo track was exported as a new MIDI file. A maximal solo length of 256 steps, which is the same as 16 bars, were allowed during training in this implementation. Therefore, if a solo exceeds 16 bars in length, the solo was sliced and exported as multiple separate MIDI tracks. Some of the songs from the dataset did not contain solos and are therefore not a part of the final dataset. The final dataset contains 382 solos from the MIDI songs. Extracting solos manually is a time-consuming task, so not all songs in the downloaded repository were used. However, songs from the corpus were mixed randomly to get a variation from all of the artists represented.

4.4. Components of the variational autoencoder

This section describes the different components of the variational autoencoder. The different components are the encoder, the latent code and two decoders, one upper-level decoder and one lower-level decoder. The architecture is inspired and shares similarities with the architecture of the VAE developed by Roberts et al. [33], but the system designed for this thesis does not use embeddings and differs significantly in all components of the VAE. The architecture in [33] is designed to generate both 2-bar loops and 32-bar melodies, while this system is designed to generate blues solos of various lengths. The VAE was implemented using PyTorch, which is a framework for implementing deep neural networks in Python with support for GPU acceleration. The architecture of the VAE can be found in figure 4.2.

4.4.1. Encoder

The full blues solo with size $sololength \times 130$ is input to a linear layer. The output of the linear layer is a lower dimensional $sololength \times 64$ matrix, which is used to initialize a single-layered LSTM network. The output size of the LSTM network equals its input size, and the output from the network is returned and used to compute $\mu_{z|x}$ and $\sum_{x|z}$.

4.4.2. Latent code

The latent distribution, z, is a Gaussian distribution with a latent dimension of 30. The latent dimensions need to be smaller than the input and output dimensions due to the detection and selection of features. The dimension of 30 was chosen based on readings [9][33][36], code examples and experiments, which confirmed that the architecture worked with the chosen latent dimension. z is of size $sololength \times 30$ which means that the z depends on the length of the solo. Longer solos will have fewer training samples that optimize the latter timesteps of the z, because solos of shorter length are unable to train the dimensions of the latent code that has more timesteps than itself. $\mu_{z|x}$ and $\sum_{x|z}$ are

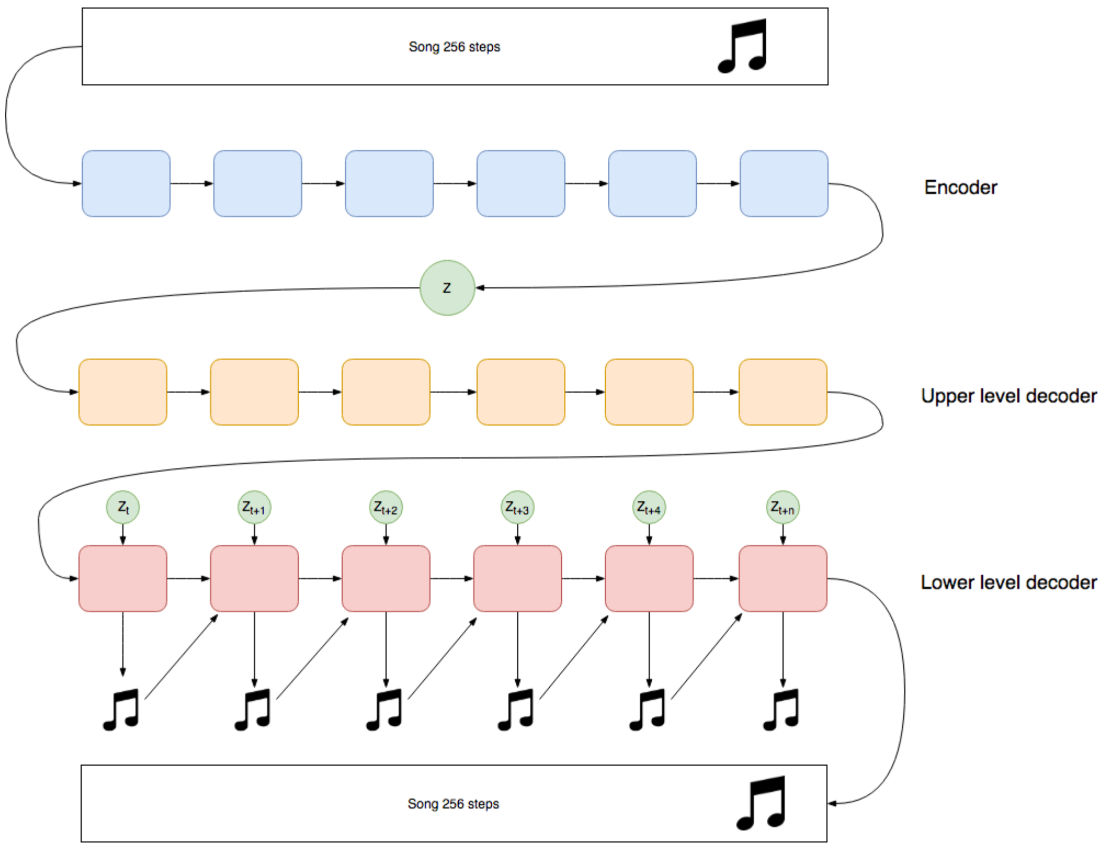


Figure 4.2.: Architecture of VAE

calculated from the output of the encoder, where the reparameterization trick described by Kingma and Welling in [20] is applied.

4.4.3. Decoder

There are two decoders, one upper-level decoder and one lower-level decoder. The upper-level decoder takes a z as input which initializes a linear layer with input dimension of $sololength \times 30$ and outputs a $sololength \times 64$ dimensional matrix, matching the hidden size. The output from the linear layer is used to initialize a single-layered LSTM network which input size equals its output size.

The lower-level decoder takes three variables as input: The previous pitch distribution output by the lower-level decoder, z_t, and the hidden state from the previous LSTM output. The initial hidden state of the lower-level decoder is initialized by the hidden state from the upper-level decoder's LSTM network to preserve the long-term dependencies. By adding an additional decoder which takes a full sample from the latent code, and using that as input to a LSTM network, the long-term dependencies of the system were increased significantly. The lower-level decoder first concatenates the previous output from the lower-level decoder and z_t. This *sololength* $\times$ 160 matrix is used as input to another single-layered LSTM network whose output initializes a linear network. The output of the LSTM network is of size *sololength* $\times$ 64. The linear layer outputs a 1×130 matrix, which is a distribution of probabilities for the different pitches.

4.5. Information flow in the variational autoencoder

This section describes the flow of the variational autoencoder during training, as well as when generating a solo.

4.5.1. Flow of training

The system is first trained with a various number of blues solos from songs that free-midi.org categorizes as blues. The preprocessed text files, that are described in 4.1, are read to numpy arrays and given to a dataloader, which is used for machine learning tasks in Python. Each of the solos is then sequentially used for training in a various number of epochs. First, the solo is fed into the encoder which is described in 4.4.1, and its output is used to sample from the from the latent code, z. After sampling z, it is used as input for the upper-level decoder described in 4.4.3, where the returned hidden state is used as the initial hidden state for the lower level decoder. Next, one probability distribution over pitches is generated at a time by using z_t, the hidden state, and the previous output. In the first iteration, the previous output is a 1×130 zero matrix, and the hidden state is the returned hidden state from the upper-level decoder. The decoder is generating one probability distribution over pitches at a time and is therefore run a number of times equal to the length of the solo. After the decoder has returned a 1×130 matrix, at each timestep the returned matrix is concatenated with the previous output for as many timesteps as the solo has. Every time the decoder returns a 1×130 matrix, the mean squared error between the output of the decoder and the current state, $state_t$, is calculated using the mean squared error definition given in equation 4.3

$$l_n = (x_n + y_n)^2 \tag{4.3}$$

After the whole solo has been decoded, the loss is returned and the latent loss is calculated. The total loss is calculated by adding the mean squared error together with the latent loss before we back propagate. The models are trained with the Adam optimizer [19] which is an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments.

4.5.2. Flow of generating a solo

After training is completed, a random sample is generated, and the sample is decoded the same way as during training. The sample from z goes first through the upper-level decoder, and the hidden state it outputs, is used to initialize the lower-level decoder. The sample has a state for each timestep, and for each timestep, that is to be generated,

$state_n$ from the sample is used as input to the lower level decoder. One probability distribution of pitches is generated at a time. The output of the decoder is a probability distribution over the pitches that are most likely to be played. After the first probability distribution of pitches is generated, it is used as input to the next decoding iteration along with the last LSTM cell's hidden state, and the sample $state_{n+1}$. This is done repeatedly until the complete solo is generated. The whole process is illustrated in figure 4.3. After all the timesteps have been successfully decoded, the concatenated output is iterated over, and the pitches that have the highest probability are selected with a few exceptions:

- A hold state cannot be chosen if there is no previous pitch, i.e., the first note to be generated cannot be a hold note.
- There is a max limit for how long a note can be held, which is 16 timesteps. If the pitch exceeds this limit, the next pitch will be a note_on pitch.

The purpose of the second point in the list above is to get variation in the generated solo and prevent extremely long notes, which are not interesting to listen to. After all the pitches have been chosen, the song is translated to a format the MIDO framework can convert to MIDI files using the MIDI decoder described in section 4.2.

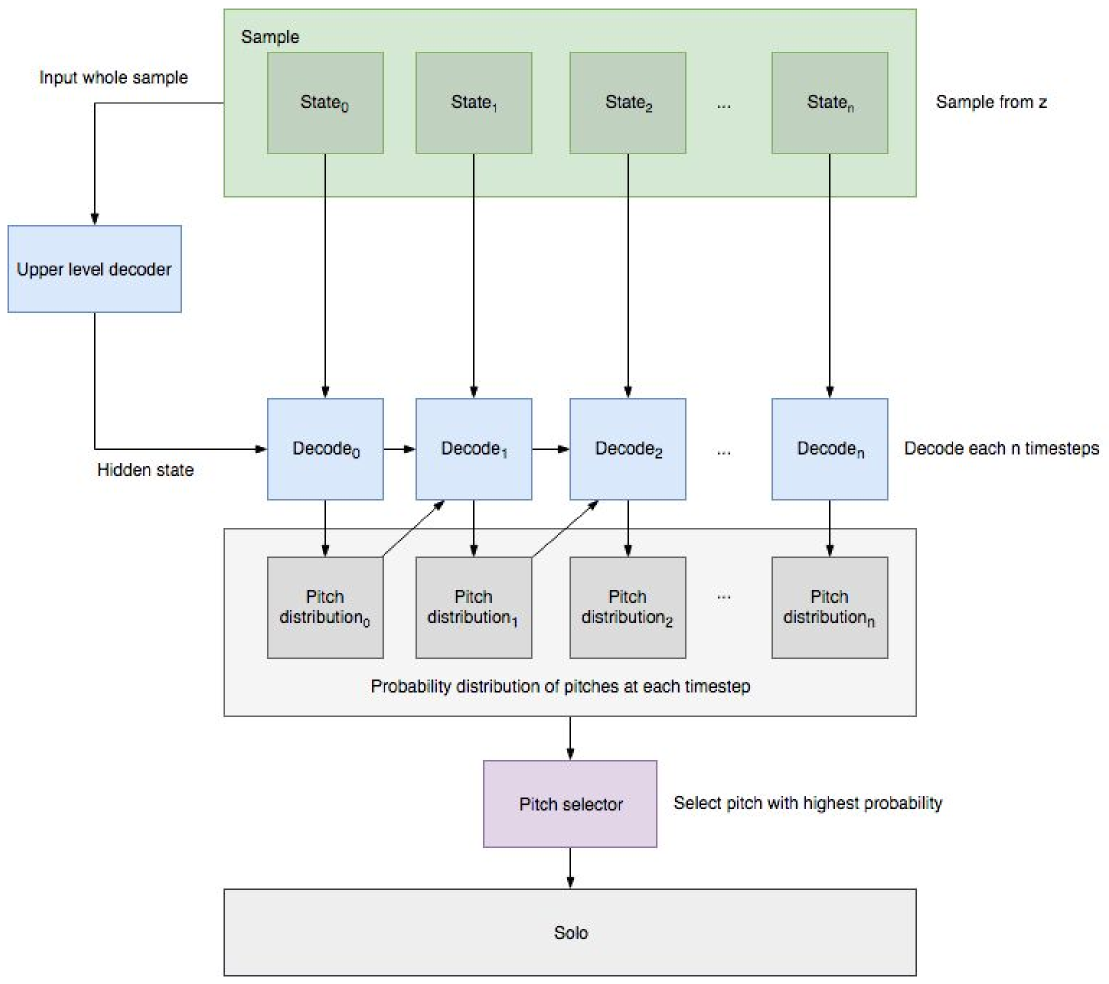


Figure 4.3.: Generate a solo from a sample

5. Experiments and Results

In this chapter, the experiments conducted are presented with the results. First, a total of three experiments for verifying the VAE are conducted, followed by experiments for generating solos and the results. Multiple solos from different models are generated over four different experiments where different values for different parameters are used. The system is evaluated through a user study and a qualitative assessment in chapter 6, and the results will be discussed further in chapter 7.

The initial implementation of the VAE was developed without systematic testing, and to verify that the network was working properly, a clean dataset was needed. Jazz music is a popular genre to experiment with within the field of computational creativity, and there is a good selection of MIDI jazz songs available. A dataset containing 20 lead melodies in the genre of jazz from the artists Art Pepper, Charlie Parker, Benny Carter and Chet Baker was used for verification. The network was trained on these 20 lead melodies for 100 epochs with a learning rate of 0.01 and produced a solo which shared similarities with the input.

However, when further inspecting the results, it became clear that the first implementation of the encoder and decoder results in big information losses that cause the autoencoder to randomly select pitches. In addition, an error with the latent code was found. To verify the errors, the VAE was trained on one jazz melody to see if it could reproduce the melody. The result was a recurrence of a single 16th note played throughout the duration of the melody, which confirms that the VAE did not work as expected.

5.1. New implementation of the Variational Autoencoder

During testing of the second version of the VAE, achieving success at each of the following three subgoals was critical in order to verify that the VAE is working properly:

1. Implement an encoder and decoder that successfully recreates a single solo without the latent code.

2. Implement the latent code and use a sample of z that is known to work (i.e., the last generated z while training).

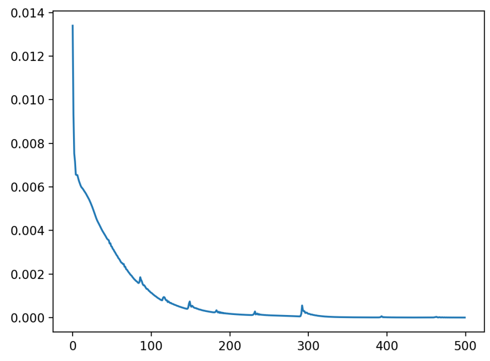
3. Sample a random z and see if it produces an output very similar to the input.

5.1.1. Step 1: Verification of the encoder and decoder

The encoder and lower-level decoder was implemented as described in section 4.4.1 and 4.4.3. The dataset in this experiment consisted of three melodies. One was a melody that contained one pitch played multiple times to ensure that the right pitch was chosen by the system, the second an arpeggio played over 8 bars, and the third eight bars of Charlie Parker's, *Scrapple From The Apple*. The first two melodies were merely used for training to quickly assess if the encoder and decoder worked. The system using only the encoder and decoder successfully recreated both the single-note melody and the arpeggio solo with a 100 % accuracy. However, these songs lacked the complicated patterns and dependencies that are found in real songs. An attempt to see if the system successfully manages to recreate *Scrapple From The Apple* which has a lot more variation in it, and 24 unique pitches, using a learning rate of 0.01 for the Adam optimizer was tried. The results are presented in table 5.1.

Epochs	Accuracy	Unique pitches
500	67.86 %	14
1000	71.43 %	21

Table 5.1.: *Scrapple from the Apple* recreated using encoder and decoder only


(a) Loss with 500 epochs

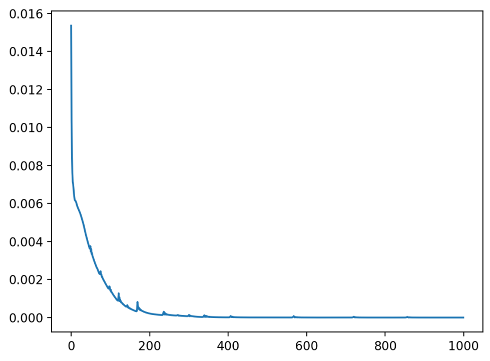

(b) Loss with 1000 epochs

Figure 5.1.: Losses with different number of epochs; encoder and decoder only

The melody is perfectly recreated at the beginning, but the system gradually introduces some errors when time passes. The conclusion of this experiment is that the system fails to capture long-term-dependencies, which call for an increase in the complexity of the system. This led to the addition of the upper-level decoder, which idea and inspiration came from Roberts et al. [33].

5.1.2. Step 2: Verification of the latent code

The latent code was implemented as described in section 4.4.2, and the upper-level decoder was also added to capture long-term dependencies. To test the latent code, the system was trained on Charlie Parker's, *Scrapple From The Apple* for 500 epochs with a learning rate of 0.01 using the Adam optimizer. On the last epoch of training, the current z was used as a sample to see if the decoder's output would be near identical to the original melody. The results were very accurate, and can be found in table 5.2.

Epochs	Accuracy	Unique pitches
500	96.42 %	24

Table 5.2.: *Scrapple from the Apple* recreated using last known z

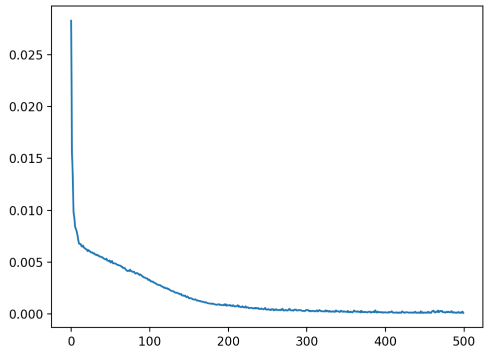


Figure 5.2.: Loss when recreating with last known z for 500 epochs

5.1.3. Step 3: Verification of variational generation

Training on only one melody for 400 epochs results in overfitting where one can expect similar output regardless of the values we choose for z. Three random samples of z were used to generate three different variations of the melodies. The results of the experiment can be found in table 5.3.

Generated solo	Accuracy	Unique pitches
1	75.89 %	19
2	69.64 %	15
3	64.28 %	17

Table 5.3.: Random samples of z for generating variations on *Scrapple From The Apple*

Looking at the results, we can see that the random sampling of z generates variations of the same melody. They are similar but have variations when sampling different values for z. This shows that the architecture for the variational autoencoder works, and successfully recreates melodies with variations.

5.2. Generation of solos

In this section the experiments conducted and the results will be presented. A series of experiments with different parameters and input was conducted to observe the behavior of the system. A total of 382 solos from what freemidi.org considers as blues were used during these experiments. There are many different parameters that need to be set for training a network and tweaking the VAE. The most important parameters are:

- Learning rate
- Dimensions of the latent code
- Number of epochs
- Number of input solos

For all of the experiments presented in the following sections, a model is trained with various values for the parameters mentioned above. In addition, three random samples from the latent code are sampled, resulting in three separate solos from each model. This allows for choosing between multiple solos, and to see variations of the same model. However, only one of the generated solos from each model is shown in the results, whichever yields the best results in terms of similarity to training data, variation, and consonance. All of the experiments were run on a GPU for increased performance.

For all of the experiments, pauses were removed from the training data. A high percentage of the training data contains a significant amount of pauses, which makes the system generate solos that contain too many pauses. In the most extreme cases, it produced solos that were entirely pauses. To avoid this, all pauses from the training data is removed in order to produce solos that only contain notes and sound more like a real solo.

5.2.1. Experiment 1: Variations on learning rate

The variational autoencoder was trained on 382 solos with different learning rates to see how the learning rate affected the system. Comparisons between the training data and the generated solos in terms of similarity using N-gram comparison and edit distance are presented for each generated solo. In addition, a comparison of variations between the training data and the generated solos is presented. The batch size was set to 1, and the LSTM networks in the encoder, upper- and lower-level decoder was set to be single-layered networks, as described in chapter 4. The system was trained for 100 epochs with a learning rate decay of $lr \times e^{-0.01}$ per epoch. The learning rate decay value was chosen with trial and error and enhanced the results of the generated solo. The generated solos for this experiment can be found in appendix A

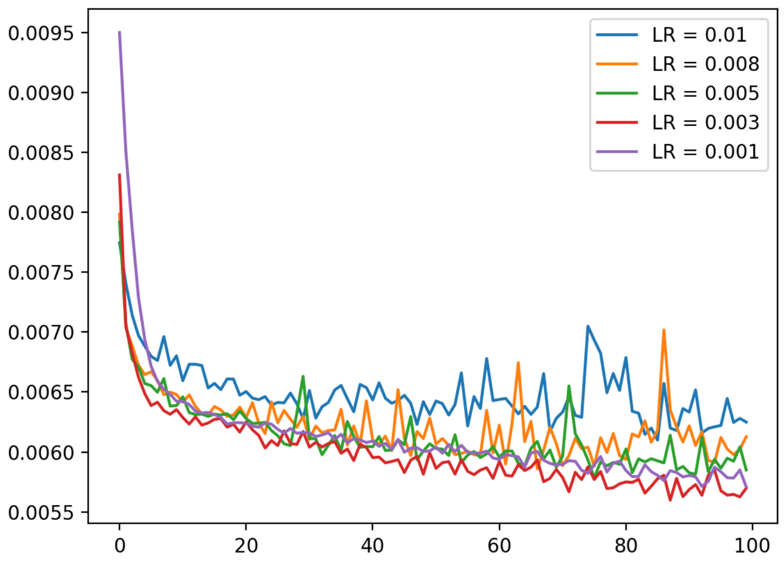


Figure 5.3.: Losses of different learning rates

In figure 5.3, the loss during training is shown. The loss after 100 epochs is lowest when training with a learning rate of 0.001 and 0.003. The highest loss after 100 epochs is with a learning rate of 0.01. Figure 5.3 also shows spikes in the loss that are most visible when training with a learning rate of 0.01, 0.008, and 0.005. This phenomenon will be further discussed in chapter 7.

Similarity

The generated solo was compared to each and every input solo. For each solo, the N-gram similarity and the edit-distance were calculated. The solos were converted to string format where the pitches played at a timestep are enclosed in brackets.

$$\textbf{Format of string: } [p_{t_1}, p_{t_2} \ldots p_{t_n}]_{t_0} + [p_{t_1}, p_{t_2} \ldots p_{t_n}]_{t_1} \cdots + [p_{t_1}, p_{t_2} ... p_{t_n}]_{t_i}$$

An example of such a string is if playing pitch 38 and 39 at timestep 1 and 2, respectively, it will generate the string: [38][39]. If pitch 39 is replaced with a hold state of pitch 38, the string would be: [38][38, 129]. In the N-gram comparisons, N was set to 3. The results are shown in table 5.4.

Parameter	LR = 0.01	LR = 0.008	LR = 0.005	LR = 0.003	LR = 0.001
Average N-gram score	0.108	0.082	0.200	0.127	0.188
Best N-gram score	0.470	0.271	0.436	0.391	0.411
Average edit distance	866.6	851.7	841.0	872.9	834.2
Best edit distance	440	409	284	256	331
Unique pitches	7	7	9	6	12

Table 5.4.: E1: Similarities with different learning rates

The average N-gram score is highest when using a learning rate of 0.005, and the same solo's average edit-distance is the second lowest. It also has the smallest best edit-distance, which means that it requires the least number of modifiable actions to be identical to an input solo. The solo generated with a learning rate of 0.001 has a very similar average N-gram score, and also has the lowest average edit-distance. The solo with the lowest average N-gram score is trained with a learning rate of 0.008, and also has the lowest best N-gram score compared to the input solos. The generated solo with learning rate 0.003, scores low on average and best N-gram compared to the other generated solos. It also has the highest average edit-distance, but the lowest best edit-distance.

Pitch variation

In figure 5.4, the results in terms of pitch variations compared to the training data are presented. The figure shows how many percent of the unique pitches in the input solos that are also found in the generated solo. The graph shows how many solos from the input data that contain the respective percentage of unique pitches in the generated solo.

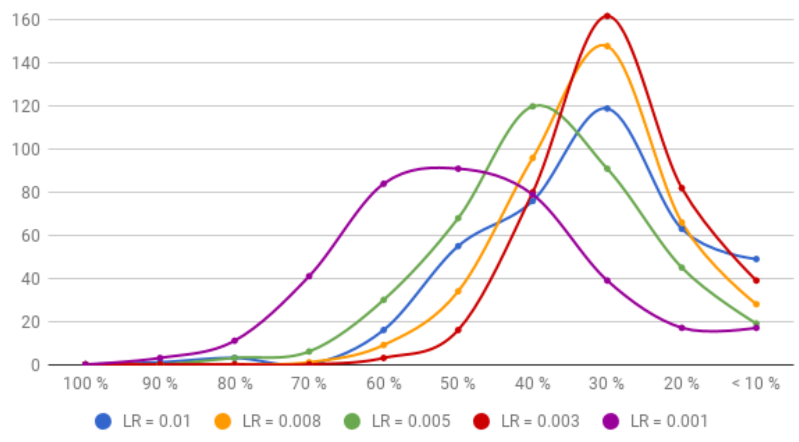


Figure 5.4.: E1: Input solos that contain the percentage of same pitches found in the generated solo

The graph in figure 5.4 shows that the learning rate of the system has an impact on the pitch variation. When using a learning rate of 0.001, a significantly larger number of input solos share more of the same pitches with the generated solos. However, the number of unique pitches found in the generated solo trained with a learning rate of 0.001, is also bigger than the rest of the solos, which will increase the probability of having a bigger percentage of the same pitches found in the input solos. The generated solo with a learning rate of 0.005 has more variation in pitches compared to the solo generated with a learning rate of 0.001, but has less variation than the others. The rest of the solos follows a very similar distribution of pitch variation.

5.2.2. Experiment 2: Variations on epochs

A series of experiments was conducted, training models for a different number of epochs. For this experiment, four different models were trained for 50, 100, 150 and 200 epochs, respectively. The purpose of this experiment is to see how the different number of epochs affect the generated solos. A learning rate of 0.005 was chosen for this experiment, with a decay of $lr \times e^{-0.01}$. The latent dimension was set to 30, and the hidden size to 64. The generated solos for this experiment can be found in appendix B.

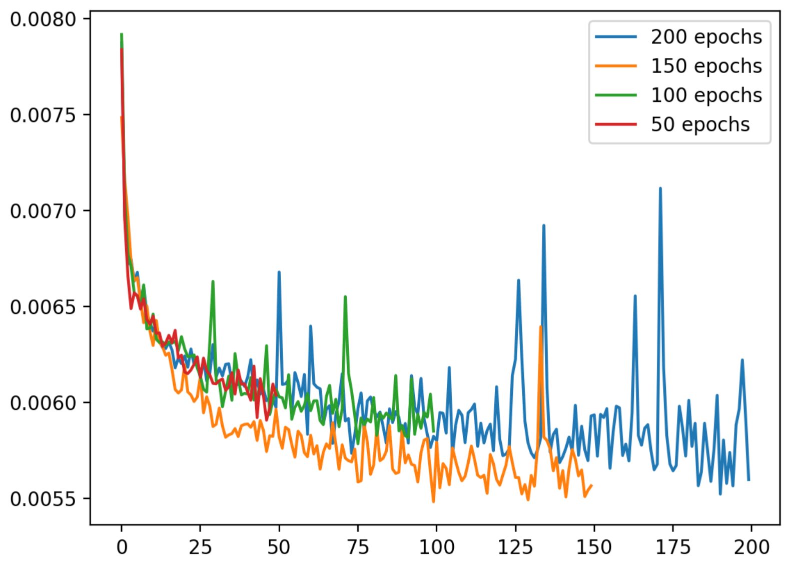


Figure 5.5.: Losses with different epochs

Figure 5.5 shows the loss when training a different number of epochs. The loss after 200 epochs is very similar to the loss after 150 epochs. The figure shows that the loss is highest when training for 50 epochs. The graph also shows that the loss has significant spikes when trained for 100, 150 and 200 epochs.

Similarity

The similarity between the training data and the generated solos is assessed in this section. It is measured the same way as in 5.2.1. The results are presented in table 5.5.

Parameter	50 epochs	100 epochs	150 epochs	200 epochs
Average N-gram score	0.078	0.200	0.108	0.049
Best N-gram score	0.314	0.436	0.323	0.139
Average edit distance	893.5	841.1	846.4	958.0
Best edit distance	425	284	363	467
Unique pitches	7	9	6	5

Table 5.5.: E2: Similarities with models trained for different number of epochs

The results show that the generated solo when trained for 100 epochs scores higher than the other generated solos in terms of similarity to the training data. The average N-gram score is almost twice as high as the generated solo which is trained for 150 epochs and gives the second highest results. This indicates that the system resembles its input best when trained for 100 epochs. It also produces solos that have more unique pitches than the others. The solo generated when trained for 200 epochs scores lower in terms of similarity, and has the least amount of unique pitches.

Pitch variation

The measures of pitch variation is the same as presented in section 5.2.1. The results of pitch variation is shown in figure 5.6.

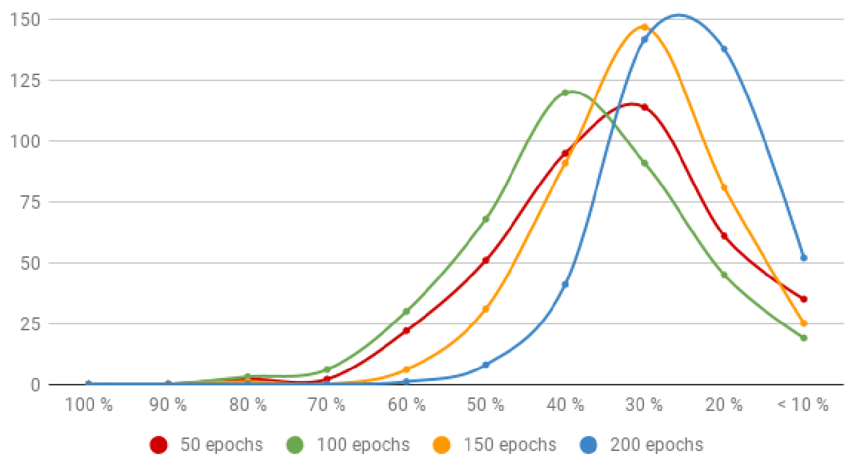


Figure 5.6.: E2: Input solos that contain the percentage of same pitches found in the generated solo

Looking at the graphs in figure 5.6, it shows that the solo generated from the model that is trained for 100 epochs share more of the same pitches with the training data compared to the rest. The solo that is trained for 50 epochs have more input solos that share a higher percentage of the same pitches than the solos trained for 150 and 200 epochs. The graph of 200 epochs shows that it shares few common pitches with most of the solos in the training data. However, the generated solo also contains very few pitches, which explains why it also has few pitches in common with the training data.

5.2.3. Experiment 3: Variation on size of training data

A series of experiments was conducted using a smaller dataset in order to see how a smaller number of input solos affected the generated solos. Different numbers of input solos were chosen for this experiment, and four different models were trained with 10, 30, 50 and 100 randomly selected solos from the complete dataset. A learning rate of 0.005 was chosen for this experiment, with a decay of $lr \times e^{-0.01}$. The latent dimension was set to 30, and the hidden size to 64. The generated solos from this experiment can be found in appendix C.

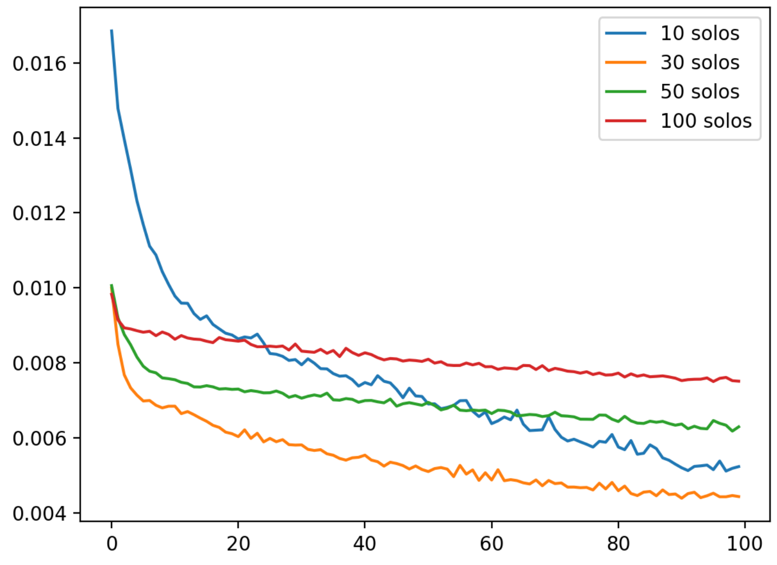


Figure 5.7.: Losses with different number of input solos

Figure 5.7 show the losses during training when the system is trained with a different number of input solos. The loss is lowest when the system is trained with 30 solos, followed by 10, 50 and 100 solos, respectively.

Similarity

The similarity in this experiment was measured the same way as described in 5.2.1. The results are presented in table 5.6.

Parameter	10 solos	30 solos	50 solos	100 solos
Average N-gram score	0.120	0.151	0.176	0.129
Best N-gram score	0.287	0.271	0.276	0.214
Average edit distance	945.3	848.9	824.4	732.96
Best edit distance	434	450	358	302
Unique pitches	12	10	8	6

Table 5.6.: E3: Similarities with different number of input solos

The N-gram measurements are increasing up towards 50 solos, but decrease when the system is trained on 100 solos. The edit distance improves with the number of solos used as training data, but the number of unique pitches found in the solos is also decreasing with the number of solos input to the system.

Pitch variation

The measures of pitch variation is the same as presented in section 5.2.1. The results of pitch variation are shown in figure 5.8.

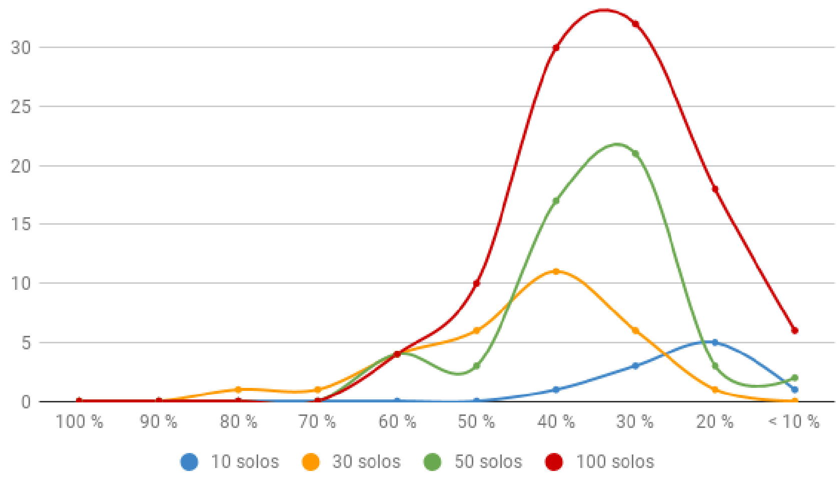


Figure 5.8.: E3: Input solos that contain the percentage of same pitches found in the generated solo

The graph shows that the solos generated perform very similar in terms of pitch variation compared to the training data. The majority of input solos share about 20 - 40 % of the pitches present in themselves with pitches present in the generated solos. The graphs are also following similar patterns, which indicates that there is a difference in pitches in the input solos and the generated solos. This means that the generated solos do not only use pitches from a single solo. The exception is the solo generated with 30 input solos, which has one input solo that shares 70 - 80 % of the same pitches as the generated solo, and one solo that shares 60 - 70 % of the same pitches.

5.2.4. Experiment 4: Changing size of latent dimension

Variation on the size of the latent dimension should affect the generated solos, and the ability the system has to detect features. Three different models where different latent dimensions of 15, 30 and 45 was chosen for this experiment. The architecture of the latent code is more thoroughly described in section 4.4.2. A learning rate of 0.005 was chosen for this experiment, with a decay of $lr \times e^{-0.01}$. The latent dimension was set to 30, and the hidden size to 64. The generated solos are found in appendix D.

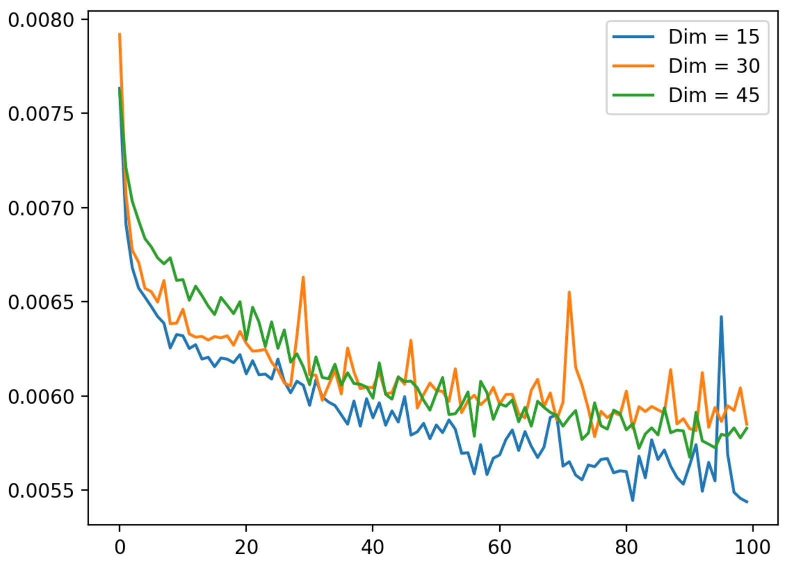


Figure 5.9.: Losses with different sizes of latent dimension

Figure 5.9 shows the loss when different sizes of the latent dimension are used. The loss after 100 epochs is lowest when trained with a latent dimension of size 15. Models with latent dimensions of size 30 and 45 have the same loss after 100 epochs. This figure shows that a latent dimension of size 15 and 30 also produces spikes in the loss.

Similarity

The results of similarity between the training data and the generated solos are presented in this section. For explanations of the terms presented in table 5.7, see section 5.2.1.

Parameter	Dim = 15	Dim = 30	Dim = 45
Average N-gram score	0.125	0.200	0.126
Best N-gram score	0.290	0.436	0.335
Average edit distance	831.8	841.0	831.8
Best edit distance	318	284	337
Unique pitches	9	9	5

Table 5.7.: E4: Similarities with different sizes of latent dimension

Table 5.7 shows that the similarity between the training data and output data is almost identical when using 15 or 45 as dimension for the latent code. The average edit distance is identical. Latent dimension of size 30 scores highest in similarity, which indicates that this configuration captures features from the training data better than the other configurations.

Pitch variation

The measures of pitch variation are the same as presented in section 5.2.1. The results of pitch variation is found in figure 5.10.

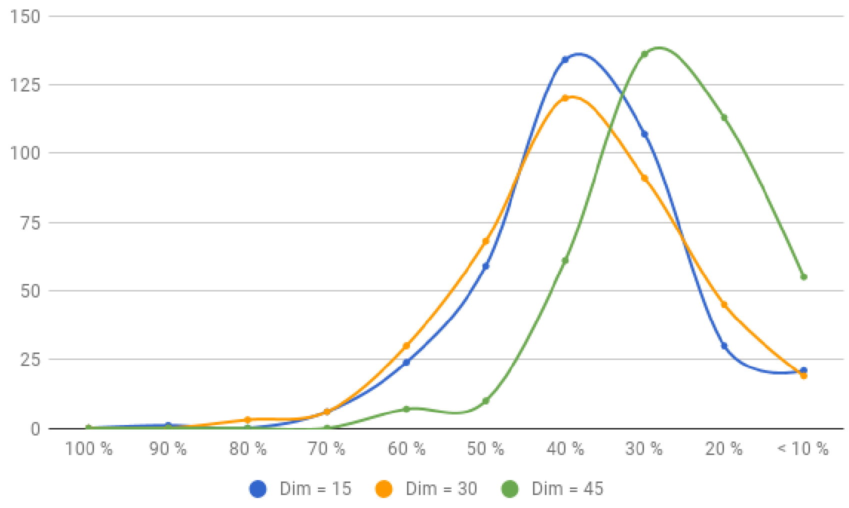


Figure 5.10.: E4: Input solos that contain the percentage of same pitches found in the generated solo

The graphs in figure 5.10 show that the solos generated from the model with a latent dimension of size 15 and 30 have very similar distributions of pitch variation compared to the input. However, the system with a latent dimension of size 30 generates a solo that share a higher percentage of the same pitches as the input solos. The solo generated with a latent dimension of 45, has fewer pitches in common with the training data, but it also has the lowest number of unique pitches of the generated solos.

6. Evaluation

In this chapter, a survey of the generated solos and the results is presented. The chapter is concluded with a qualitative assessment of some of the generated solos.

6.1. Survey

The purpose of the experiment is to observe how the solos sound to human ears and to measure how good a group of people perceives different aspects of the solos. The group consisted of 14 people which mainly identified themselves as hobby artists or music enthusiasts. They were asked to listen to 16 solos where some are generated by the system, and some are real solos from real songs used as training data for the system. 9 of these solos were generated by the system, while 7 were randomly selected solos from the training set. Backing tracks were added to all of the solos to provide context, and consisted of a guitar track that played a chord progression over the solo and a simple drum beat. The chord progressions played in the background were customized to each and every solo, to ensure that the backing tracks were in the same key as the solo, but kept very basic to minimize the influence the backing track may have on the solos. The chord progressions were played with a clean guitar MIDI instrument in $\frac{1}{2}$ notes or $\frac{1}{4}$ notes. For the drum tracks, notes were either $\frac{1}{4}$ notes or $\frac{1}{8}$ notes, and variations on hi-hats were added as well. Some solos have a drum track with open hi-hat, some with closed and others with ride cymbal. The evaluation was presented online where the people that evaluated the solos were provided with a link where they could listen to the solos and then answer questions about them. The same questions were asked for all 16 solos. The group had to answer the following questions:

Q1: Do you think the solo is generated by the AI?

Q2: What genre do you think the solo is?

Q3: How repetitive do you think the solo is?

Q4: How interesting do you think the solo is?

Q5: How original do you think the solo is?

Q6: How surprising was the solo?

Q7: How ‘good’ do you think the solo is?

6.1.1. Results of Q1 and Q2

Table 6.1 shows the results of **Q1** and **Q2**. It shows how many people that thought the respective solos were generated, real, or was unable to tell. It also shows the genre the majority perceived the respective solos to be. The real solos are marked grey in the table, while the generated are white.

Solo	Generated	Real	Unable to tell	Genre
1	5	8	1	Pop
2	8	3	3	Rock
3	8	4	2	Rock
4	3	8	2	Pop
5	4	8	2	Blues
6	13	1	0	Rock
7	3	9	2	Pop
8	12	0	2	Rock/blues
9	10	0	4	Pop/rock
10	2	11	1	Blues
11	2	9	3	Rock
12	10	4	0	Rock
13	12	1	1	Rock
14	2	7	5	Pop/rock/blues
15	7	3	3	Rock
16	12	1	1	Rock

Table 6.1.: Results of Q1 and Q2

In figure 6.1 the average scores of **Q2 - Q7** on all real and generated solos are presented. The scores of the real solos will act as a baseline for the generated solos. In the following figures, the scores of each generated solo are shown, and a baseline is presented in the same figure. The baseline is the average score of the real solos in the respective measurement. In addition, the scores of the extremities of the real solos are shown.

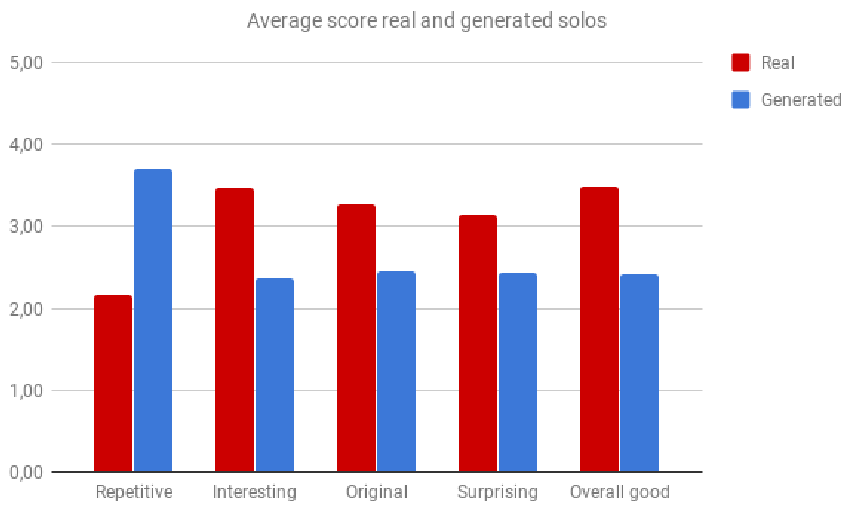


Figure 6.1.: S1: Average scores real and generated solos

6.1.2. Results Q3: Repetitiveness

Q3 was how repetitive the listeners perceived the generated solos, and the results are presented in figure 6.2. The orange line marks the score of the real solo that was most repetitive, while the grey line marks the score of the real solo that was least repetitive. The red line indicates the baseline, which is taken to be the average value for the real solos.

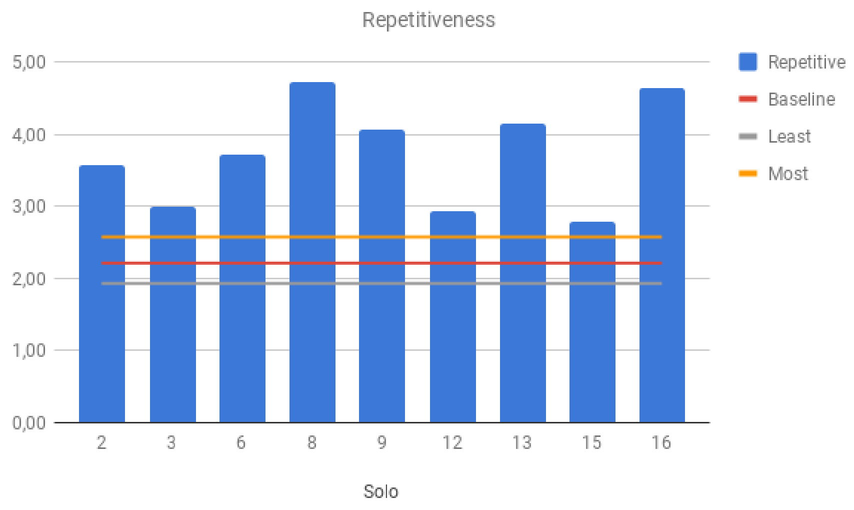


Figure 6.2.: S2: Repetitiveness of the generated solos

All of the generated solos are perceived more repetitive than all the real solos. A few of the solos are approaching the baseline, while the rest are perceived vastly more repetitive than the real solos. Solo 3, 12 and 15 is very close the same score as the lowest scoring real solo, which indicates that the participants of the survey perceived these solos almost as repetitive as some of the real solos.

6.1.3. Results Q4: Interesting

Figure 6.3 shows how interesting the solos were perceived to be by the listeners. The orange line marks the score of the real solo that was most interesting according to the participants of the survey, while the grey line marks the score of the real solo that was least interesting. The red line indicates the baseline , which is taken to be the average value for the real solos.

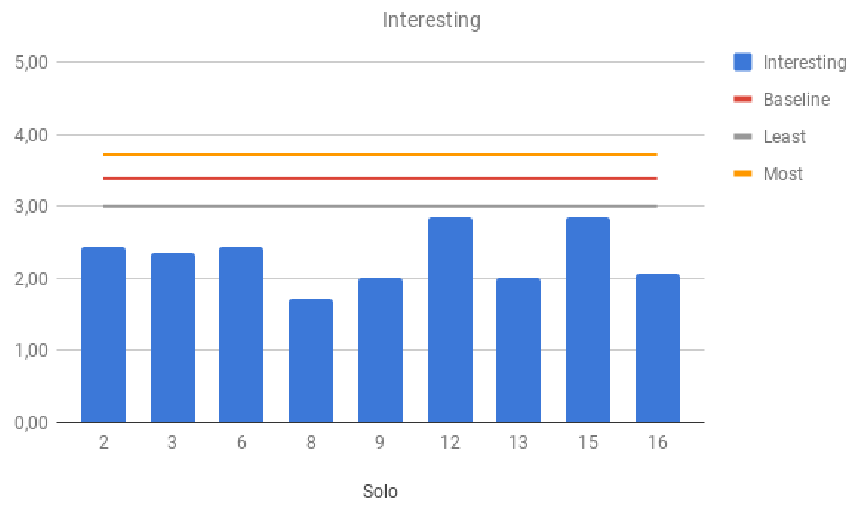


Figure 6.3.: S3: How interesting the generated solos was perceived

Solo 12 and 15 are very close to being as interesting as the lowest scoring real solo. Solos 2, 3 and 6 perform very similar and scored slightly lower than solos 12 and 15. The solos that got lower scores in terms of repetition are also perceived as more interesting to the listeners.

6.1.4. Results Q5: Originality

Figure 6.4 shows how original the solos were perceived to be by the listeners. The orange line marks the score of the real solo that was most original according to the participants of the survey, while the grey line marks the score of the real solo that was least original. The red line indicates the baseline, which is taken to be the average value for the real solos.

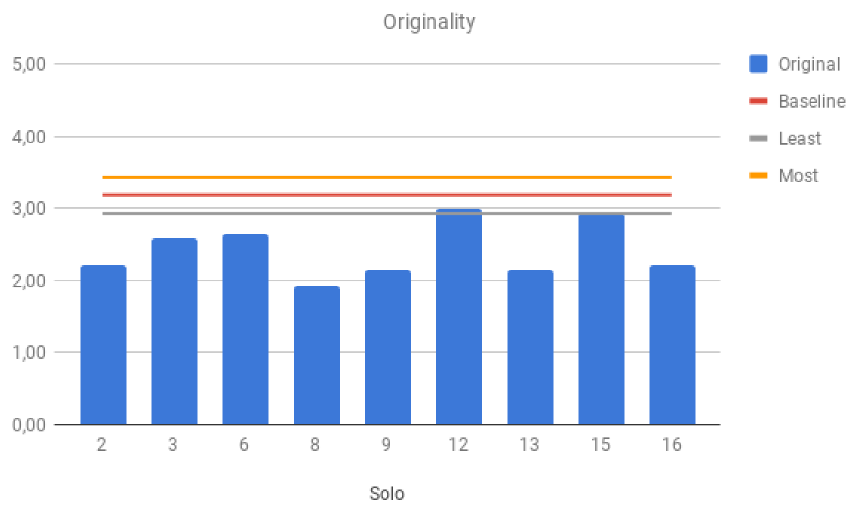


Figure 6.4.: S4: Originality of the generated solos

Solo 12 and 15 are also doing well in terms of originality. Solo 12 outperforms the lowest scoring real solo, while solo 15 scores the same as the lowest scoring real solo. Both are very close to the baseline, which indicates that the participants of the survey think of these solos as being as original as the training data. Solo 3 and 6 also scores relatively high in this measure, and both are approaching the score of the real solo that is considered least original. The rest of the solos are outperformed by the real solos.

6.1.5. Results Q6: Surprising

Figure 6.5 shows how surprising the solos were perceived to be by the listeners. The orange line marks the score of the real solo that was most surprising, while the grey line marks the score of the real solo that was least surprising. The red line indicates the baseline, which is taken to be the average value for the real solos.

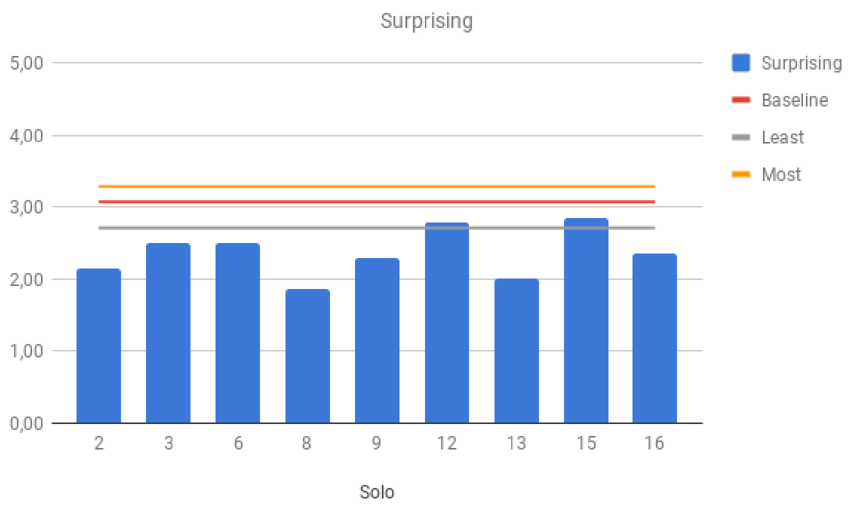


Figure 6.5.: S5: How surprising the generated solos were to the listeners

The scores in this measure are almost identical to the scores of originality, which can indicate that the questions are strongly correlated. Solo 12 and 15 both outperform the least surprising real solo, and both are very close to the baseline. Solo 3 and 6 scores similar, and both are close to the lowest scoring real solo.

6.1.6. Results Q7: Overall

Figure 6.6 shows how ‘good’ the listeners perceived the solo overall. The orange line marks the score of the real solo that was considered the best overall, while the grey line marks the score of the real solo that was considered the worst overall. The red line indicates the baseline, which is taken to be the average value for the real solos.

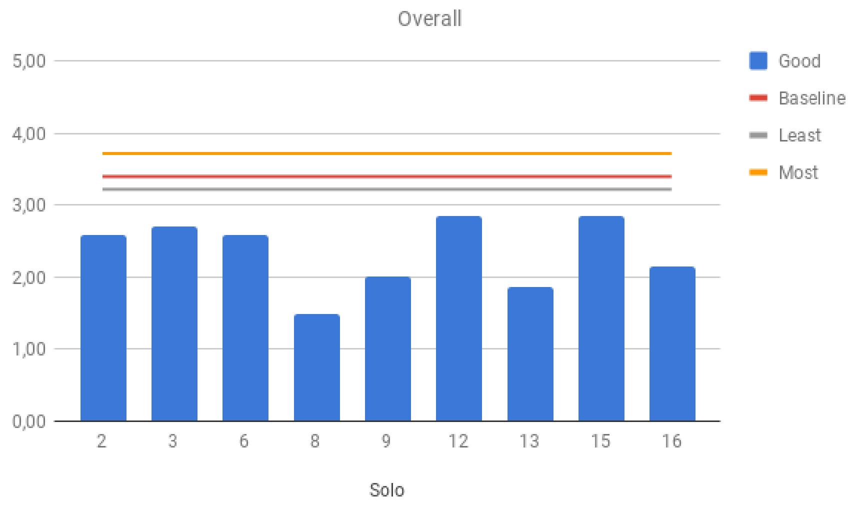


Figure 6.6.: S6: How the generated solos were perceived overall

Overall, solo 12 and 15 scores the best, but are closely followed by solo 2, 3 and 6. Looking at figure 6.6, we can see that the generated solos are generally perceived to sound worse than the real solos. However, solo 12 and 15 nearly score the same as the lowest scoring real solo, closely followed by solo 3, 6 and 2. The rest of the solos score significantly worse.

6.1.7. Overview over generated solos

The generated solos used in this survey are mostly the same as the ones in experiments 1-4 in the previous section, with the exception of one generated solo which was generated earlier. An overview over the solos are found in table 6.2, which includes the parameters of the solos, as well as what experiment in section 5.2 they were a part of.

Solo # in survey	Experiment	LR	Epochs	Number of input solos	Lat dim
2	1	0.01	100	382	30
3	1	0.005	100	382	30
6	3	0.005	100	30	30
8	3	0.005	100	100	30
9	4	0.005	100	382	45
12	-	0.005	100	382	30
13	3	0.005	100	10	30
15	1	0.001	100	382	30
16	2	0.005	50	382	30

Table 6.2.: Overview of solos in the survey

6.2. Qualitative assessment

In this section, a qualitative assessment of the solos is presented. The solos are assessed by Magnus Sahlgren, a computer linguist and guitarist at the Swedish Institute of Computer Science. Sahlgren was the lead guitarist of the band Lake of Tears, and has also played in bands such as Dismember and Celeborn. 6 solos were sent to him to get a deeper more qualitative assessment on the solos generated.

His comments on the solos are the following:

- The solos are easily playable, by using a guitar technique called *tapping*. *Tapping* is a technique where the player is using both hands on the neck of the guitar and taps with the hand which usually is used for strumming on the fretboard to play a note.
- Sahlgren thought the solos were repetitive and often used octaves. It is for playing these octaves the tapping technique could be used. He is not sure if these solos qualify as blues, but rather a form of jazz.
- Qualitatively, he does not think the solos are very good, and he thinks that the solos sound like someone who just discovered tapping, but did not learn to play yet.

He thinks the solo from *Experiment 3* which is trained on 30 solos was interesting. He thinks it sounds like the AI is playing something "wrong" in this solo, which made him wonder about the training data. The solo uses the Neapolitan scale in the key of E.

7. Discussion

In this chapter, the results of the experiments are discussed. First, the creativity of the system is discussed in terms of variation and originality, followed by a section on how realistic the solos are. Next, the goals of the thesis are discussed, before the chapter is concluded with answers to research questions.

7.1. Computational creativity

When evaluating how creative a system is, the three processes that define creativity discussed in section 2.6 are used. The three processes are:

- Generation
- Variation
- Originality

However, no further detail on the generation part of will be discussed in the evaluation, because the process is about bringing something into being, which is true for all of the experiments when the system successfully generated solos in every experiment.

7.1.1. Variation

The system does generate solos that have variation from the input data, and it is able to generate variations of itself, in terms of generating variants of a solo with different sample values from the latent distribution from the same model. The output of the system is highly affected by the different parameters in the VAE, and the results show that the right values and combinations of the parameters produce solos that share some similarities with the training data. The learning rate of the system impacts the loss during training, and we can see from experiment 1 that the solo with the lowest loss, do not generate the solo most similar to the input data as one could expect. There can be multiple reasons for this, but an obvious one is that only three random samples from the latent distribution were extracted. This leaves large parts of the latent space unexplored, and it may be that the right sample from the latent distribution would generate a solo that is more similar to the output. Overfitting can also be an explanation for this problem. From experiment 2, we see that when increasing the number of epochs, the solos generated have fewer unique pitches. Based on this, it is likely that the system learns to prefer only a small selection of pitches, which are played more often throughout the solo than other pitches. This will lead to solos with little variation and repetitive patterns with very few unique pitches, which may also lead to low scores in the similarity measurements.

Solo 12 and 15 scored highest in the overall measurement in the survey. Although solo 12 was not in either of the experiments in section 5.2, the model the solo is generated from is trained with the same parameters as one of the solos in *experiment 1*, and the results show these solos also scored highest in terms to similarity to the training data. This is logical because the training data is extracted solos from real songs. If the system generates solos that are similar to real solos, it is sensible that they will sound good to the human ear as well.

Comparisons between results from the survey and results in pitch variation show that there is a correlation between how many pitches the training data share with the generated solos and how good the generated solos sound to people. The results in pitch variation from *experiment 1* show that the generated solos from the system when trained with a learning rate of 0.01, 0.005 and 0.001, had a higher percentage of real solos from the training data that share more unique pitches with the generated solos than the other two learning rates. The results from the survey reveal that the same solos also sound better to the people surveyed. When inspecting the solos that scored low in the survey, the same solos also have a high pitch variation from the training data, which supports the assertion that the more similar the solos are to the training data, the better they sound to the human ear.

The N-gram comparison measures similarity on a scale of 0 to 1, where a score of 1 means that the two strings are identical. Considering that the highest average N-gram score was 0.2, it would be interesting to see if it is possible to generate a solo that had a higher N-gram comparison score and see if the output solo scored higher in the survey. However, an increase in the N-gram score may result in a loss in creativity and lead to solos that are too similar to the training data. One could think that the number of solos used for training theoretically would affect the N-gram score positively, because with fewer solos, there is less variation in the training data, and therefore it should be easier for the system to learn. However, looking at the results of experiment 3, we can see that this is not the case. In fact, the N-gram score increases when the number of input solos are increased. There can be various reasons for this. The input solos were chosen randomly, and it may be that the 10 input solos chosen as input varied significantly, and were in different keys and styles. If we look at the training losses from the different experiments, we can see that the solos that scored highest in terms of similarity also had a somewhat high loss compared to the other solos.

The variational autoencoder does generate solos with significant variation when sample different values for z. Looking at the results from section 5.1, we see that the VAE is able to produce variations of the same solo when trained on only one solo. During the solo generation experiments, three random samples from z were taken, and a solo was generated for each sample where all had some variation from each other. The amount of variation between the solos varies on a sample by sample basis.

7.1.2. Originality

Originality needs to have an effect of surprise, and it comes in three different degrees, which are individual, social and universal. Given that the system is trained with solos from real songs, one might expect to get somewhat similar sounding solos, which makes it unlikely to produce solos of universal originality. Based on the fact that a large part of the people that participated in the survey categorized themselves as 'music enthusiasts', it might be possible to trigger some sort of social originality if they are inexperienced to blues, but due to the number of people that participated, it can only be categorized as individual originality. It is hard to measure originality because what people perceive as originality is individual. However, an attempt to measure it from the participants of the survey was made. The results in originality show that one solo outperformed a solo from the real dataset, while another scored the same as the real solo that was considered least original. This shows that the system is able to generate solos that perform better than a real solos in terms of originality. The same two solos are also very close to the baseline and are considered to have an average originality, with scores around 3 out of 5. This indicates that these solos are considered to be somewhat original, and as original as the training data surveyed. Originality is hard to measure in a quantitative survey, but based on the results, people seem to think that there is some originality to some of the generated solos and that some are as original as the training data. As previously mentioned, originality is 'the effect of surprise'. When asked how surprising the generated solos sounded, the participants of the survey responded almost identical as they did in originality measurement.

7.2. Realism

From the results of the survey, we see that the system is able to generate solos that perform close to the real solos. In some of the aspects, the generated solos outperform the real. In terms of repetitiveness and how interesting the solos are perceived, the real solos outperform the generated. However, some of the generated solos score very close to the lowest scoring real solo in both measures. This indicates that the participants of the survey think some of the generated solos are almost as interesting, and have nearly the same amount of repetition as some real solos. Looking at originality and surprise, we can see that the system generates solos that in certain cases may even outperform a real solo. In addition, some of the solos in these measurements also approach the baseline. This indicates that participants of the survey not only think that these generated solos are more original than the lowest scoring real solo, but they also perform close to the average score of all real solos.

The generated solos are generated with different models and parameters, which leads to some sounding better than others. The results from table 6.1 show that there is some disagreement about which solos that are generated and which that are not. There is some confusion regarding the generated solos 2, 3 and 15 on whether they are generated or not, which shows that the system is capable of generating solos that can be mistaken for real solos. However, the majority of the participants were correct on every single solo, which shows that people generally can distinguish the generated solos from the real ones.

The survey revealed that there might be some limitations to the dataset. As previously mentioned, there are many different artists from different decades that are represented in the training data, which might not be what the participants associate with blues. When looking at table 6.1, it becomes clear that the participants did not consider the training data as blues. The backing tracks added to the solos likely affected these results as well. Simple chord progressions of four chords are often found in pop and rock music, while blues often has other more characterizing chord progressions like 12-bar blues or 8-bar blues. Even though only seven solos from the training data were surveyed, it is an indication that what freemidi.org considers blues, is not what the participants of the survey consider blues. Only two of seven solos from the training set have fallen in the category of blues.

Some of the generated solos do score near the baseline in most measurements and may outperform a real solo in terms of originality and how surprising the solo is perceived. This shows that with the right configuration of the VAE, the system is able to produce solos that may perform as good as real solos in some areas, and can cause confusion of the reality of the solo.

The qualitative assessment indicates that the solos are not qualitatively considered very good in the ears of a musician. However, the solos are playable on a real guitar using a common guitar technique called tapping. Similar to the participants of the survey, Sahlgren was not sure if he would call the output blues, but he considered the solos to be rather more like jazz. One of the solos did invoke some sort of feeling of surprise for Sahlgren, where he perceived the AI to play "wrong", which may be perceived as creativity from the system's perspective.

7.3. Goals

In this section, the goals of this thesis that were presented in the introduction are discussed. These goals are:

G1: Develop the necessary tools to extract musical information from MIDI and use it to train a network

G2: Develop a variational autoencoder that generates blues solos

7.3.1. G1: Develop the necessary tools to extract musical information from MIDI and use it to train a network

The system is able to successfully read a MIDI file and encode it into a format that can be used as input to the network. The format of the musical representation for the neural network was implemented the same way as in [33], where they also successfully generate melodies with a variational autoencoder. When taking the encoded MIDI file, and feeding it into the MIDI decoder, the system successfully recreates the MIDI file to its original state with a 100 % accuracy. Looking at the experiment where the VAE is verified, we can see that the MIDI file is successfully formatted to a format that the system is able to interpret, and that can be used to train the network to recreate a melody with high accuracy. However, the MIDI encoder does not extract all musical information found in the MIDI file. Information like pitch bending and pitch velocity is not encoded into the neural network format, which limits the system's ability to capture all features of a solo. Even so, the MIDI encoder's ability to encode such information was intentionally left out of the system, due to the complexity such features introduce and is left for future work.

7.3.2. G2: Develop a variational autoencoder that generates blues solos

The results show that the design of the architecture is capable of capturing features from training data, and successfully generate solos that have variation from the training data. The experiment of verifying the VAE proved that the system is capable of learning long-term dependencies from training data and recreate a solo with high accuracy. In the same experiment, we can also see that the system generates variations of the training data, based on different values for z, which verifies that the implementation of the latent distribution was successful. The training loss of the experiments revealed some interesting information. The training data consists of many different artists that play music in different styles, which impacts the loss during training. The network is trained with a batch size of 1 and backpropagates for every solo. This has an impact on the gradient descent when solos that are very different are trained after each other and could cause sudden increases in the loss. If the batch size were to be increased, the loss would be an average over all the solos in the batch, which would decrease the spikes in the loss, as we can clearly see happen in figure 5.5. If we sample from the latent distribution and generate a solo from the VAE at a time where we have one of these spikes in contrast to right before or after, the output could turn out significantly different. This affects the stability of the system and should be addressed in future work.

The survey revealed that the system is able to generate solos that perform similarly to some real solos. This indicates that the latent code of the VAE learns features from the training data, which enables the system to generate solos that to the participants of the survey sound similar to the real solos. However, the survey also revealed that the training data itself was not considered blues by the participants, and one can not expect the output to be considered as blues if the training data is not considered as blues. The system also has its limitations in terms of being unable to handle important features of blues music. Pitch bending and microtones have a huge influence on blues music, and this system does not consider such features. Blues solos generally contain a great deal of pitch bending, and there are common patterns in solos that are played repeatedly where one or more pitches played in the pattern are pitch bent. When listening to the output, one can hear such patterns occuring in the generated solos, but when the pitch bending is removed, all that is left is something that sounds very repetitive, and the system fails to incorporate the bluesy feeling in the generated patterns. Solos that are played by humans do not have perfect timings, which adds a certain feel to a solo. When quantizing the data, imperfections from the dataset get removed, which leads to solos that have perfect timings. When doing so, a feature that would make the solos seem more realistic and human vanishes. Lastly, the system does not consider pauses, which again leads to losses in features from the training data. The initial implementation did include pauses, but there was no penalty or control over the pitch prediction, which led the system to generate solos that had an undesirable amount of pause notes.

7.4. Answer to research questions

In the following subsections answers to the research questions discussed in chapter 1 are given.

7.4.1. R1: Can the generated solos from the system be mistaken for a solo from a real song?

The quality of the solos generated by the system differs greatly. Solos are generated from models with different parameters and configurations in the different experiments, which as discussed earlier results in solos of different quality. The quality of the solo will influence the likelihood it has to be mistaken for a real solo, and the survey revealed that people generally can tell if a solo is generated or not. However, when looking at solo 2, 3 and 15 in table 6.1, almost half of the people asked are wrong, or say they are unable to tell if the solo is generated by the AI or not. This shows that the generated solos can be mistaken for a real solo.

7.4.2. R2: What factors have an effect on the networks ability to generate solos that are more similar to its input?

As discussed earlier, experiments have shown that different values of different parameters have an impact of the system's ability to generate solos that are similar to the input data. However, when inspecting the results, it becomes clear that most factors impact the systems ability to produce similar data. In experiment 1-4, the results show that the chosen learning rate, number of epochs, size of training data, and the size of the latent dimension all affect the systems ability to generate solos more similar to the training data. Values of different parameters need to be chosen carefully in order for the system to best imitate the training data. The architecture of the system will also heavily impact its ability to produce similar output. As shown during verification of the VAE, when introducing the second decoder to the system, its ability to handle long-term dependencies increased.

8. Conclusion and Future Work

In this chapter, the conclusion of the thesis is presented along with future work. The future work section addresses limitations of the system that should be improved in the future.

8.1. Conclusion

A variational autoencoder that generates solos based on solos from real songs was implemented. Software for extracting, and encode musical data from MIDI files to a format that can be interpreted and used as input to a neural network, as well as a decoder to turn the encoded data back to MIDI were successfully developed. The architecture of the VAE has proven able to capture long-term dependencies in solos, and successfully generate solos that differ from the input. In terms of generating solos in the style of blues, the system has its limitations. Mainly because the survey uncovered that for the most part, the training data is not considered blues. A more consistent training set that is considered blues, and with less variation, is needed in order for the system to generate solos that are also considered as blues. In addition, core elements of blues such as pitch bending, velocity and pauses are removed from the training data. The implementation of these features would most likely improve the system's ability to generate blues solos vastly. The generated solos vary significantly from the training data, and an interesting observation is that the solos that got higher scores in terms of its similarity to the training data, generally sounded better to the people that listened to the solos. The results from the survey also revealed that some of the generated solos are approaching the baseline, and may even outperform some real solos in certain aspects. This shows that the system is able to produce solos that are considered by some people to be as 'good' as some of the real solos. Some of the generated solos are also close to the baseline in terms of originality, and are considered to be somewhat original. With the generation of a solo, variation from the training data, and the output being thought of as somewhat original, Robinson's three processes of creativity are fulfilled. Based on this definition of creativity, the system can generate solos that are creative.

8.2. Limitations and future work

The system itself is verified, and the experiments show that it works as expected. However, the spikes in the loss during training reveal that the stability of the system has room for improvement. These spikes could cause a significant difference in the output if a solo is generated at a time where such a spike occurs. This problem could be addressed by increasing the batch size as previously discussed. While this issue could have a great impact on the output of the solos, other limitations that directly affect the system's ability to produce better blues solos should, in my opinion, be addressed first. The most significant limitations are:

L1: The system is unable to process string bending and pitch velocity in MIDI files.

L2: By quantizing the data, we remove human feeling which is an important aspects of blues solos.

L3: It lacks functionality to handle pauses in solos.

L4: It should have some restrictions on repetitiveness.

L5: The training data is mostly not considered blues by the people surveyed.

Possible future work for this system, is to address these limitations, which would improve the realism of the system, and quality of the generated solos.

8.2.1. Pitch bending and pitch velocity

This definitely is the most significant feature which would increase the realism of the solos generated. String bending and microtones are a big part of blues, and when removed, a great deal of musical data is also lost. String bending is encoded as a series of MIDI-messages where each MIDI message contains information of how much a current pitch should increase or decrease its frequency. When a series of these messages are sent sequentially after each other, it sounds like a smooth increase in pitch, just like when bending a string on a guitar. However, it is hard to encode this data into the neural network because the values of the pitch increase are continuous, and would result in millions of combinations as input dimension. A solution could be to have different states of pitch bending, where there are small discrete values possible for pitch bending, e.g., a semitone or a whole tone bending. After the network has predicted a pitch that should be bent, it could be added a series of MIDI-messages that handle pitch bending after the *note_on* message has been appended to the MIDI-messages list. This would increase the complexity of both the MIDI encoder and decoder vastly and is therefore left as future work.

8.2.2. Feeling

This can be solved by adding a random factor to the time component in the MIDI decoder. However, this is a quick fix that will not learn from the input data and detect common *off patterns* in solos. With *off patterns*, we mean patterns of imperfect timings that occur regularly. This should be looked further into in order to increase the feeling and realism of the generated blues solos.

8.2.3. Pause handling

To handle the pause problem, the pauses were simply removed from the training data before training the system. This should be improved in the future so that the system can generate solos that can predict pauses. As of now, none of the solos generated have any pauses in them. Many blues solos contain a significant amount of pauses, and when we remove pauses from the training data, we also remove some core elements from the blues solos.

8.2.4. Repetitiveness

Many of the generated solos contain a great deal of repetition, which could make the solos sound uninteresting at times. However, repetition is a core element of a solo, but it is important that it does not get overwhelmingly much. Future work of the system should be to address this issue. This could be solved by introducing a type of penalty which penalizes the prediction of the same pitch n number of times after each other.

8.2.5. Improve training data

The survey revealed that most of the training data was not considered as blues to the participants. This clearly affects the system's ability to generate solos in the genre of blues. Time should be spent finding more sophisticated ways to make sure that the training data really is blues. At the very least, the current dataset should be classified in sub-genres with respect to time of release.

Bibliography

[1] Poulter Albert. ngram 3.3.2. [Online] Available: `https://pypi.org/project/ngram/`, 2018. Accessed 11/06/18.

[2] Chip Bell. Algorithmic music composition using dynamic markov chains and genetic algorithms. *Journal of Computing Sciences in Colleges*, 27(2):99–107, 2011.

[3] John A Biles. Genjam: A genetic algorithm for generating jazz solos. In *ICMC*, volume 94, pages 131–137, 1994.

[4] John A Biles. Interactive genjam: Integrating real-time performance with a genetic algorithm. In *ICMC*, 1998.

[5] Jean-Pierre Briot, Gaëtan Hadjeres, and François Pachet. Deep learning techniques for music generation-a survey. *arXiv preprint arXiv:1709.01620*, 2017.

[6] Gino Brunner, Yuyi Wang, Roger Wattenhofer, and Jonas Wiesendanger. Jambot: Music theory aware chord based generation of polyphonic music with lstms. *arXiv preprint arXiv:1711.07682*, 2017.

[7] Carl Doersch. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*, 2016.

[8] Douglas Eck and Juergen Schmidhuber. A first look at music composition using lstm recurrent neural networks. *Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale*, 103, 2002.

[9] Otto Fabius and Joost R van Amersfoort. Variational recurrent auto-encoders. *arXiv preprint arXiv:1412.6581*, 2014.

[10] Serena Yeung Fei-Fei Li, Justin Johnson. Generative models. [Online] Available: `https://www.youtube.com/watch?v=5WoItGTWV54`, 2018. Accessed 15/01/18.

[11] Robert Fisher. What is creativity. *Unlocking creativity: Teaching across the curriculum*, pages 6–20, 2004.

[12] Howard Gardner. Fostering diversity through personalized education: Implications of a new understanding of human intelligence. *Prospects*, 27(3):346–363, 1997.

[13] Khan Academy Gerard Schwarz. Music basics. [Online] Available: `https://www.khanacademy.org/humanities/music/music-basics2/notes-rhythm/a/glossary-of-musical-terms`, 2018. Accessed 19/06/18.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[15] Gaëtan Hadjeres and François Pachet. Deepbach: a steerable model for bach chorales generation. *arXiv preprint arXiv:1612.01010*, 2016.

[16] Jay A Hennig, Akash Umakantha, and Ryan C Williamson. A classifying variational autoencoder with application to polyphonic music generation. *arXiv preprint arXiv:1711.07050*, 2017.

[17] John R Hershey and Peder A Olsen. Approximating the kullback leibler divergence between gaussian mixture models. In *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, volume 4, pages IV–317. IEEE, 2007.

[18] Robert M Keller and David R Morrison. A grammatical approach to automatic improvisation. In *Proceedings, Fourth Sound and Music Conference, Lefkada, Greece, July.*, 2007.

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

[20] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[21] Bill Lucas. Creative teaching, teaching creativity and creative learning. *Creativity in education*, pages 35–44, 2001.

[22] M Marques, V Oliveira, S Vieira, and AC Rosa. Music composition using genetic evolutionary algorithms. In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, volume 1, pages 714–719. IEEE, 2000.

[23] Jon McCormack. Grammar based music composition. *Complex systems*, 96:321–336, 1996.

[24] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[25] midi.org. Front page. [Online] Available: `https://www.midi.org`. Accessed 05-03-2018.

[26] midi.org. Midi history:chapter 6-midi is born 1980-1983. [Online] Available: `https://www.midi.org/articles/midi-history-chapter-6-midi-is-born-1980-1983`. Accessed 22-03-2018.

[27] James Anderson Moorer. Music and computer composition. *Communications of the ACM*, 15(2):104–113, 1972.

[28] Michael C Mozer. Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multi-scale processing. *Connection Science*, 6(2-3):247–280, 1994.

[29] Daniel Müllensiefen, Klaus Frieler, et al. Cognitive adequacy in the measurement of melodic similarity: Algorithmic vs. human judgments. *Computing in Musicology*, 13(2003):147–176, 2004.

[30] Andrew Ng. Logistic regression as a neural network. [Online] Available: `https://www.coursera.org/learn/neural-networks-deep-learning/lecture/Z8j0R/`, 2018. Accessed 10/01/18.

[31] Christopher Olah. Understanding lstm networks. [Online] Available: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`, 2018. Accessed 06/03/18.

[32] Dolmetsch Organisation. Time signatures and meter. [Online] Available: `https://www.dolmetsch.com/theoryintro.htm`, 2018. Accessed 22/06/18.

[33] Adam Roberts, Jesse Engel, and Douglas Eck, editors. *Hierarchical Variational Autoencoders for Music*, 2017.

[34] Ken Robinson. *Out of our minds: Learning to be creative.* John Wiley & Sons, 2011.

[35] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[36] Alexey Tikhonov and Ivan P Yamshchikov. Music generation with variational recurrent autoencoder supported by history. *arXiv preprint arXiv:1705.05458*, 2017.

[37] Bernard Widrow and Michael A Lehr. 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, 78(9):1415–1442, 1990.

[38] Terry Winograd. Linguistics and the computer analysis of tonal harmony. *Journal of Music Theory*, 12(1):2–49, 1968.

Appendices

A. Generated solos from Experiment 1

A.1. Generated solo with learning rate of 0.01

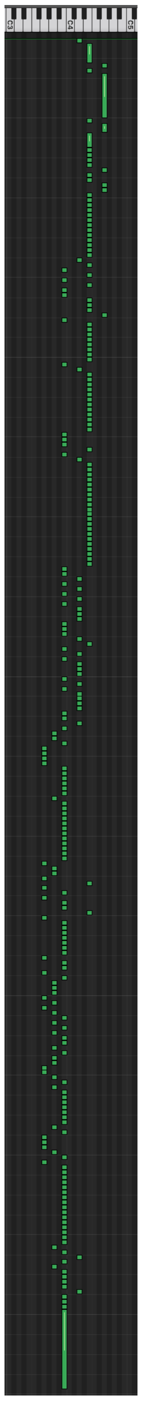


Figure 1.: Generated solo with 0.01 as learning rate

A.2. Generated solo with learning rate of 0.008

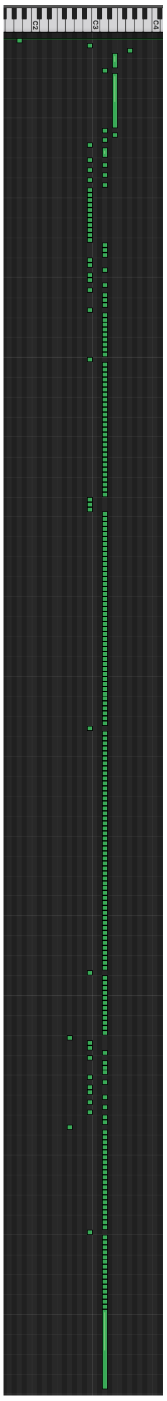


Figure 2.: Generated solo with 0.008 as learning rate

A.3. Generated solo with learning rate of 0.005

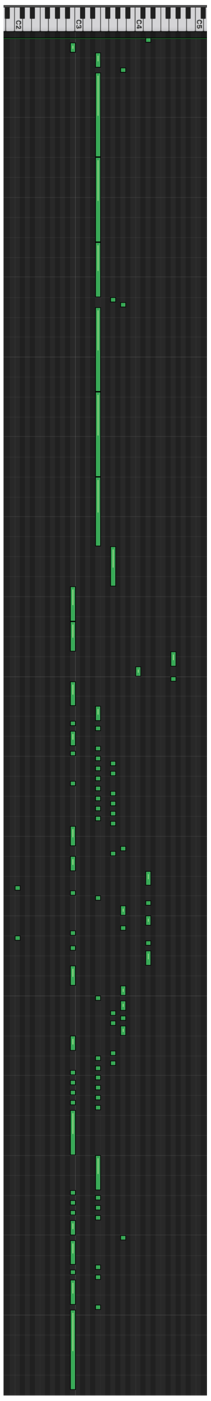


Figure 3.: Generated solo with 0.005 as learning rate

A.4. Generated solo with learning rate of 0.003

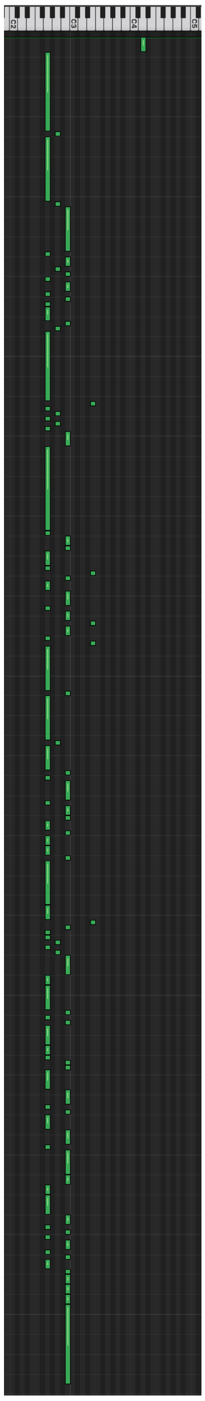


Figure 4.: Generated solo with 0.003 as learning rate

A.5. Generated solo with learning rate of 0.001

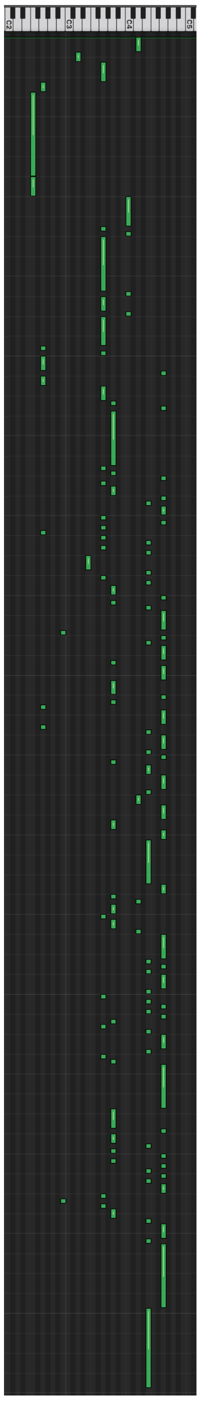


Figure 5.: Generated solo with 0.001 as learning rate

B. Generated solos from Experiment 2

Note that the solo generated with 100 epochs is the same as in A.3.

B.1. Generated solo when trained for 50 epochs

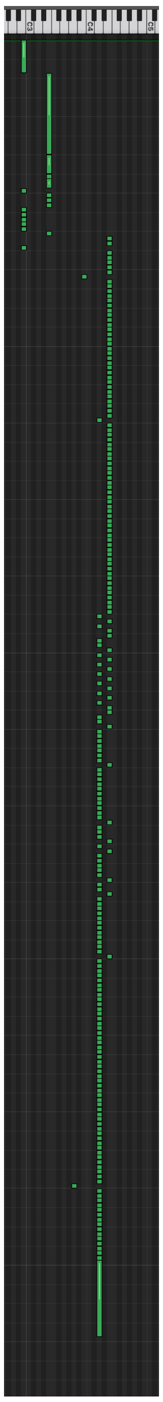


Figure 6.: Generated solo when trained for 50 epochs

B.2. Generated solo when trained for 150 epochs

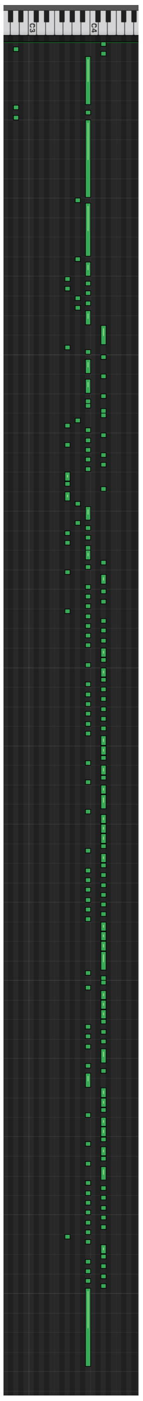


Figure 7.: Generated solo when trained for 150 epochs

B.3. Generated solo when trained for 200 epochs

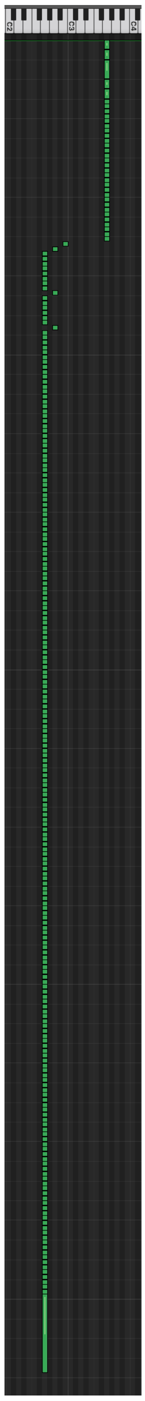


Figure 8.: Generated solo when trained for 200 epochs

C. Generated solos from Experiment 3

C.1. Generated solo when trained with 10 input solos

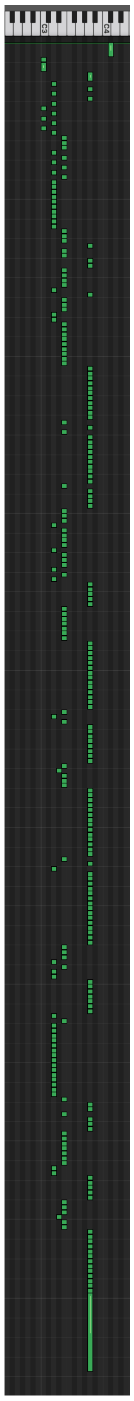


Figure 9.: Generated solo when trained with 10 input solos

C.2. Generated solo when trained with 30 input solos

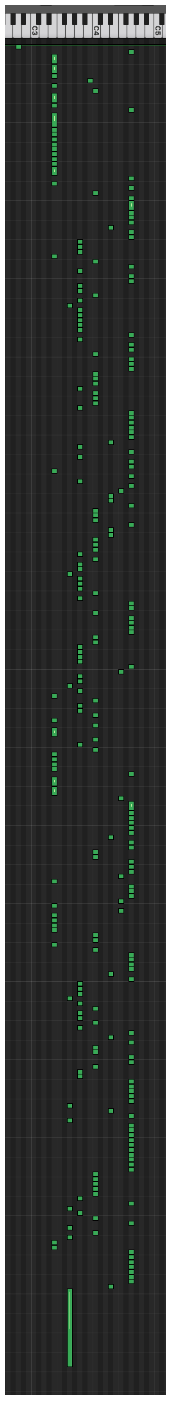


Figure 10.: Generated solo when trained with 30 input solos

C.3. Generated solo when trained with 50 input solos

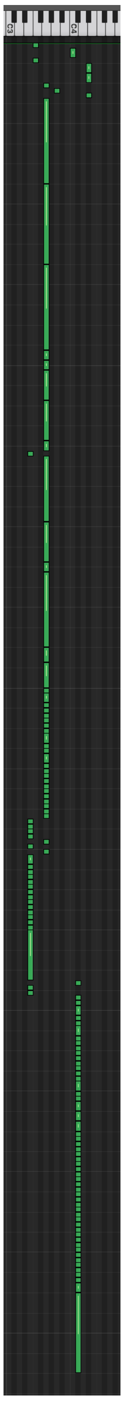


Figure 11.: Generated solo when trained with 50 input solos

C.4. Generated solo when trained with 100 input solos

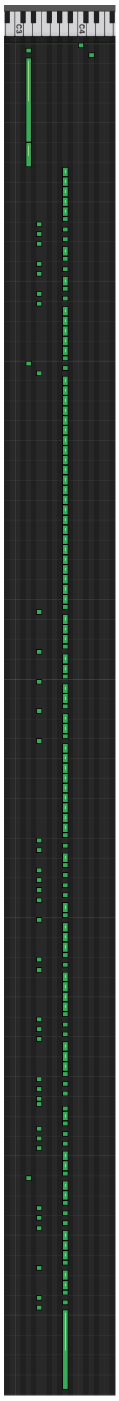


Figure 12.: Generated solo when trained with 100 input solos

D. Generated solos from Experiment 4

Note that the solo generated with a latent dimension of size 30 is the same as in A.3.

D.1. Generated solo when trained with latent dimension of size 15

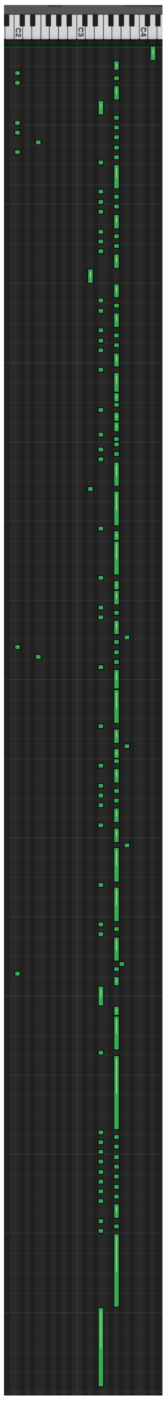


Figure 13.: Generated solo when trained with a latent dimension of size 15

D.2. Generated solo when trained with latent dimension of size 45

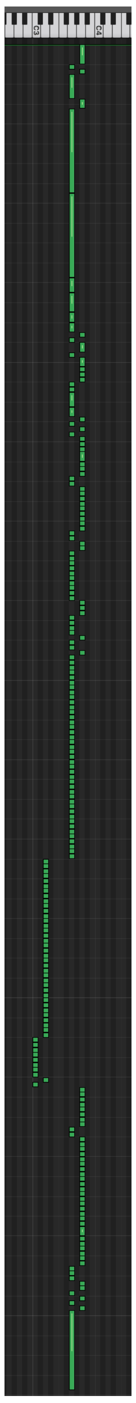


Figure 14.: Generated solo when trained with a latent dimension of size 45