Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Endre Valstad Berg

Automatic Accompaniment Generation Using Generative Adversarial Networks

Master's thesis in Computer Science
Supervisor: Björn Gambäck
June 2020

NTNU
Norwegian University of Science and Technology

Endre Valstad Berg

Automatic Accompaniment Generation Using Generative Adversarial Networks

Master's Thesis in Computer Science, Spring 2020

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 presents research into the use of generative adversarial networks consisting of recurrent neural networks to generate musical accompaniment for melodies.

Through a series of experiments, the capabilities of the architecture with regards to the task were explored. The overall results presented limited capabilities of generating fitting accompaniments for a leading melody, though some desirable behaviour was exhibited from certain configurations of the architecture. Six different experiments were conducted. The first experiment was a baseline configuration that generated accompaniments that showed no variations across all generations, instead outputting the same single note for all input melodies. The second experiment implemented techniques to handle this problem, but ultimately produced the same results. The third experiment tested the learning rates, and experiment four and five tested the number and size of the input time steps. These three experiments generated accompaniments that contained some degrees of variation, though they all were very similar. The final experiment altered the architecture so that the generated output was only based on the leading melody, and not on the previously generated outputs as had been the case in the preceding experiments. This configuration displayed a greater degree of variation across the generated accompaniments, and exhibited behaviour that the system adjusted its output based on the events in the leading melody. The main limitation across all experiments is the small vocabulary of notes that the system obtains. This leads to the generated accompaniments all containing the same few notes which causes the generated accompaniments to rarely fit with leading melody, as they are usually in different keys.

Overall the capabilities of the architecture displayed in this thesis are limited, but the architecture does display some promising behaviour which should motivate further research. This research should focus on increasing the vocabulary of notes that the system possesses, and giving information to the system as to which notes fit together.

Sammendrag

Denne masteroppgaven presenterer forskning på bruken av generative adversarial networks bestående av recurrent neural networks til å generere akkompagnement for melodier,

Gjennom en rekke eksperimenter ble arkitekturens evne til å gjennomføre denne oppgaven utforsket. Reultatene viste en begrenset evne til å generere passende akkompagnement for en melodi, men det ble også demonstrert noen ønskede kvaliteter fra visse konfigurasjoner av arkitekturen. Seks forskjellige eksperimenter ble gjennomført. Det første eksperimentet var en basiskonfigurasjon som genererte akkompagnement som ikke demonstrerte noen grad av variasjon. I stedet inneholdt de genererte akkompagnementene kun den samme enkelte noten uavhenging av input til systemet. Det andre eksperimentet implementerte teknikker som håndterte dette problemet, men resultatene ble fortsatt de samme. Det tredje eksperimentet testet forskjellige læringsgrader, og eksperiment fire og fem testet antallet og størrelsen på tidsstegene som ble brukt som input. Det siste eksperimentet endret på arkitekturen slik at genereringen kun var avhengig av melodien, og ikke de tidligere genererte tidsstegene slik som i de foregående eksperimentene. Dette demonstrerte mer variasjon i de genererte akkompagnementene, og systemet demonstrerte atferd som tydet på at det tilpasset genereringen basert på hendelser i melodien. Hovedbegrensningene til systemet gjennom alle eksperimenter er det få antallet noter i vokabulæret systemet oppnår. Dette fører til at alle genererte akkompagnement inneholder de samme notene, noe som gjør at de genererte akkompagnementene sjeldent passer til melodien, da de vanligvis er i forskjellige tonearter.

Systemet demonstrerte begrensede evner gjennom denne oppgaven, men det viste noe lovende atferd som bør motivere videre forskning. Denne forskningen bør fokusere på å utvide vokabulæret av noter som systemet oppnår, og gi systemet informasjon om hvilke noter som passer sammen.

Preface

This Master's Thesis is submitted to the Norwegian University of Science and Technology (NTNU) as part of the requirements for the degree of Master of Science in Computer Science. The work on this thesis was conducted at the Department of Computer Science, NTNU, Trondheim. This thesis was supervised by Professor Björn Gambäck.

I would like to thank my supervisor Björn Gambäck for all the help and guidance he has provided throughout the process of this Master's Thesis. I would also like to thank Dr. Marinos Koutsomichalis for the feedback he has given during our video conferences.

Endre Valstad Berg
Trondheim, 23rd June 2020

Contents

List of Figures

List of Tables

1. Introduction

Computational creativity is a field consisting of a wide variety of different topics, where the goal is to replicate creative tasks such as painting, poetry and music using a computer. Colton and Wiggins (2012) define computational creativity as "The philosophy, science and engineering of computational systems which, by taking on particular responsibilities, exhibit behaviours that unbiased observers would deem to be creative".

In this thesis the focus is on *computational musical creativity*. It builds upon work done during a previous project, which explored the research done in the field, with a focus on using artifical intelligence (AI) for automatic accompaniment generation (Berg, 2019). The findings of that project presented an architecture using deep learning, which aims to generate accompaniment for a given melody. This thesis implements and tests this architecture, which uses recurrent neural networks with generative adversarial training in order to be capable of generating accompaniments for melodies. The data used for training this model is a collection of 8-bar phrases, where the guitar part is used as the melody, and the system is trained to generate an accompanying bass part.

This chapter presents the background and motivation for the thesis, as well as the research goals and questions that will be answered. The final two sections presents the research contributions of the thesis, along with a general overview of the thesis structure.

1.1. Background and Motivation

Pieces of music generally consist of more than just a melody played by a single instrument. Often there are distinct parts being played by different instruments, all combining into creating the music pieces we know and love. This thesis focuses on automatic accompaniment generation, which means generating parts that fit with a given melody.

1.2. Goals and Research Questions

Goal 1 *Explore the capabilities of Generative Adversarial Network consisting of Recurrent Neural Networks in the field of automatic accompaniment generation*

Using generative adversarial networks (GAN) to generate accompaniments for a melody has previously been researched. These implementations typically use convolutional neural networks (CNN) to generate the accompaniment. The goal of this thesis is to use a GAN consisting of recurrent neural networks (RNN) in order to generate accompaniments. This approach is inspired by the architecture C-RNN-GAN by Mogren (2016), in which a GAN consisting of RNNs is trained to generate melodies.

Research question 1 *How do the generated artefacts compare to the training data?*

In order to be able to accurately evaluate the architecture, there needs to be some form of evaluation metric. An objective evaluation metric would be to look at some of the key statistics of the generated accompaniments, and comparing them to the statistics of the accompaniments in the training data.

Research question 2 *Can the generated artefacts be considered valid accompaniments for the leading melody?*

Comparing the generated accompaniments to the real accompaniments is not enough to conclude whether a generated accompaniment fits with a leading melody. There can be many valid accompaniments for each melody. In order to judge whether accompaniments fits with the leading melody or not, a number of key properties of the generated accompaniments will be evaluated. These include that the notes in the accompaniment fits with the notes of the leading melody, that the generated accompaniment reacts to changes in the leading melody and the overall sound of the accompaniment and leading melody together. For the different experiments conducted, these features will be evaluated in order to answer whether the generated artefacts are valid accompaniments for the input melody.

Research question 3 *How do the different system configurations affect the generated accompaniments?*

Training a GAN requires a great deal of optimization and finely tuned hyperparameters in order for it to be able to accomplish the task that it is set out to do. Therefore a series of experiments will be conducted which will test how different configurations of the model affect the generated accompaniments the model produces.

1.3. Contributions

The following are the main contributions from this Master's Thesis:

1. *An architecture consisting of recurrent neural networks using generative adversarial training that displays some desirable behaviour for generating accompaniments.*
2. *An evaluation of the generated accompaniments, which highlights the system's capabilities and limitations.*
3. *An evaluation of how different configurations affect the generated accompaniments.*
4. *Proposal for future research involving the architecture presented in this thesis.*

The source code for the architecture is available here:
`https://github.com/endrevb/AccompGANiment`

Samples of the generated accompaniments and leading melody, as well as the real accompaniments for all experiments can be found here:
`https://drive.google.com/drive/folders/17xlDJhI4idSmaXopGrj9W0VVs5_oZVe_?usp=sharing`.

1.4. Thesis Structure

Chapter 2 contains the relevant background theory for this thesis. The history of computational musical creativity and automatic accompaniment generation, as well as the state-of-art, are described in chapter 3. Chapter 4 presents the proposed architecture, and gives an overview of the training data. Chapter 5 presents the different experiments conducted. In chapter 6, the results of the system are evaluated and discussed. Finally, chapter 7 presents the conclusion, as well as the future work.

2. Background Theory

In order to be fully able to understand the contents of this thesis, some background theory is required. Since the subject of this thesis is related to both artificial intelligence and music, concepts and terms from these fields will be used. The first section presents some relevant musical terminology and concepts. After this, the relevant theory behind MIDI and AI are presented. Parts of this section is taken from the exploratory project which preceded this thesis (Berg, 2019).

2.1. Music Theory

Western music is based around a twelve-tone system. Usually, a western music piece consists of melody, harmony and rhythm played by different instruments.

Pitch is a distinct frequency of sound. We denote these pitches using the first seven letters of the alphabet, as well as ♯ and ♭[1]. We may also use a number after this to denote the *octave* of the note. An octave is, as defined by Schwarz (2018), an interval between two notes, where the upper note has twice the pitch of the lower note. For example, A4 has a pitch of 440 Hz, with A3 being 220 Hz.

The symbol ♯ after a letter means that the note that is to be played is one semitone above the note indicated by the letter in the notation. For example, F♯ is one semitone above F (Zeitlin and Goldberger, 2002). In the same manner ♭ means that the note is to be played one semitone below.

Tonic is also known as the root. It is the note that indicates which key the music piece is in. Musicians often refer to the tonic as "home", since playing the tonic often feels like a resolution to a musical phrase (Levine, 1995). The tonic is what is called the first scale degree, with the second note in the scale being the second scale degree and so on.

Scale is a subset of the twelve tones. For example, we have the A major scale which consists of the notes A, B, C♯, D, E, F♯, G♯[2]. As we can see the A major scale consists of seven notes. Different scales contain different notes, as well as a different number of notes. For example, the minor pentatonic scale often found in rock and blues only contains five notes. It is possible for two scales to contain the same notes. An example of this is the A major and F♯ minor scale, who share the exact

[1] In some European regions, natural B is referred to as H, and natural B♭ is referred to as B.
[2] This paper will not distinguish between flats and sharps, so E♭ equals D♯ for example.

same notes. One thing to note here is that the tonic of each scale is different (A for A major and F♯ for F♯ minor). In fact, all the scale degrees of the notes are different between the scales. This is called a *relative minor* (inversely, a relative major). There also is something called *parallel major* and *parallel minor*, in which the scales share the same tonic (C major and C minor).

Key is the set of pitches, and their corresponding chords, in which a musical piece revolves (Schonbrun, 2012). A key can be either major or minor, so a song in the key of A major will revolve around the seven notes, and the accompanying chords, of the A major scale. It is also possible to play notes not in the key, as displayed for example in jazz and blues with the "blue" note (Levine, 1995). Schonbrun (2012) writes that "keys are the DNA of music".

Sheet music is a method of sharing musical information. A piece of sheet music contains a visual representation of the musical information. This makes it easier for musicians in an ensemble to play a music piece for example. The annotated sheet music tells the musicians what and when they are going to play, so that the musicians don't need to memorize a whole piece of music. It also speeds up the learning process, as it is possible to play the piece of music without being taught what to play first. Sheet music is also used as a teaching tool to teach different musical concepts. This is how it will be used in this thesis. Figure 2.1 shows the sheet music for two octaves of the C major scale. Each note has a distinct vertical placement, which makes the sharing of musical information simple and intuitive.

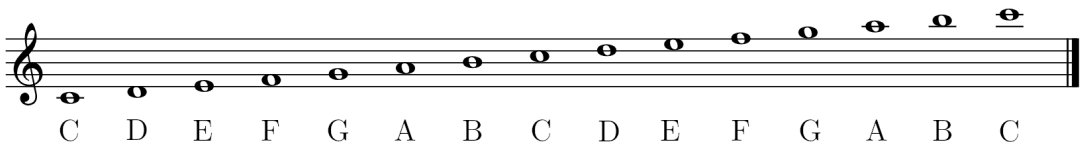


Figure 2.1.: Two octaves of the C major scale.

Bar is also called a measure, and is the distance between two bar lines (Zeitlin and Goldberger, 2002). A bar can contain a number of notes and pauses. The notes and pauses have a duration denoted by the symbol they are represented with in the sheet music. This duration is based on the bar, so a sixteenth note (𝅘𝅥𝅯) has the duration $\frac{1}{16}$ of the current bar.

Note symbol	Pause symbol	Duration
𝅝	𝄻	$\frac{1}{1}$
𝅗𝅥	𝄼	$\frac{1}{2}$
♩	𝄽	$\frac{1}{4}$
♪	𝄾	$\frac{1}{8}$
𝅘𝅥𝅯	𝄿	$\frac{1}{16}$

Table 2.1.: Note and pause symbols, and their duration.

Time signatures is a way of denoting the number of beats in a bar. The most commonly used time signature is $\frac{4}{4}$, but other time signatures are more prevalent in less mainstream music genres such as jazz, waltz and progressive rock (Levine, 1995). $\frac{4}{4}$ represents that there are four beats in the bar, which means that a bar can consist of one whole note, two half notes, four quarter notes etc. (Zeitlin and Goldberger, 2002).

Intervals are the relationships between notes. Figure 2.2 shows the intervals between the tonic and the other notes in C major, and the difference in intervals in C minor (Schonbrun, 2012). These intervals are the building blocks of harmonies and chords (Nettles, 1987).

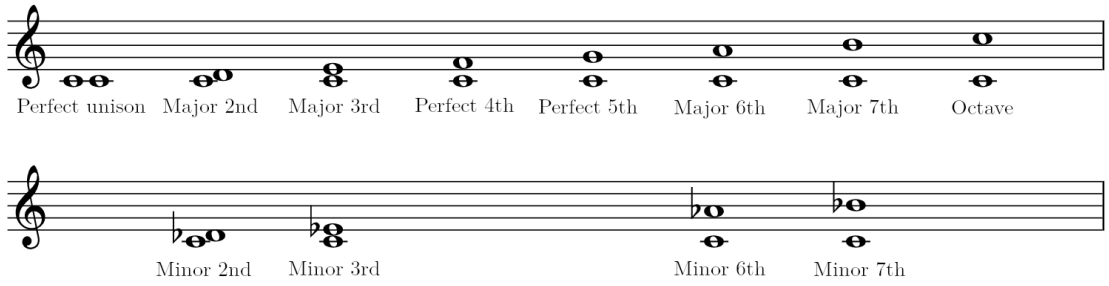


Figure 2.2.: Intervals in C major and C minor.

Chord is the result of putting together several intervals. The most common chords are the major and minor chords, which consists of a tonic, a third and a fifth. These types of three note chords are called triads (Nettles, 1987). A perfect fifth can also be viewed as the major third above the third of the tonic. So in essence, triads are just thirds stacked upon each other. As seen in figure 2.2, the C major and C minor scales have a different third interval where C major has a major third and C minor

has a minor third. This means that a C major chord contains a major third, while a C minor chord contains a minor third. The third is what defines a chord as either major or minor, since both major and minor chords share the same tonic and fifth.

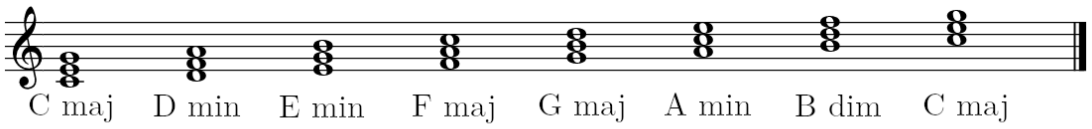


Figure 2.3.: Chords in C major.

Figure 2.3 shows the chords of the C major scale. Maj and min indicate whether the chord is major or minor. B dim is what is called a diminished chord, which means that it has a flat fifth instead of the perfect fifths of major and minor chords. So a diminished chord consists of a minor third and a flat fifth. A flat fifth is the minor third above the third of the tonic, i.e. a perfect fifth lowered one semitone. Diminished chords are what is called *dissonant*. Levine (1995) writes that dissonance provides tension, and sets up resolution. Music without dissonance could quickly become dull and uninteresting, while music consisting only of dissonance would be unpleasant and annoying. Dissonance is a tool that can be used to create dynamics in music pieces, keeping them from being stale and boring.

Harmony is a set of intervals played at the same time (Rich, 2019). Also, harmony often refers to a melody line that accompanies a main melody. A simple example of this is harmonizing using thirds. This is done by moving the note that is currently being played in the main melody up three scale degrees in the appropriate scale, i.e. a third.

Figure 2.4.: Harmony using thirds in C major.

Figure 2.4 shows that the harmony is following the melody line. If a second harmony playing fifths is added we get triad chords.

Piano-roll representation is a way of representing notes in a 2-dimensional space where duration is horizontal and the pitch is vertical. Figure 2.5 shows an example of the piano roll representation for the harmonization in figure 2.4.

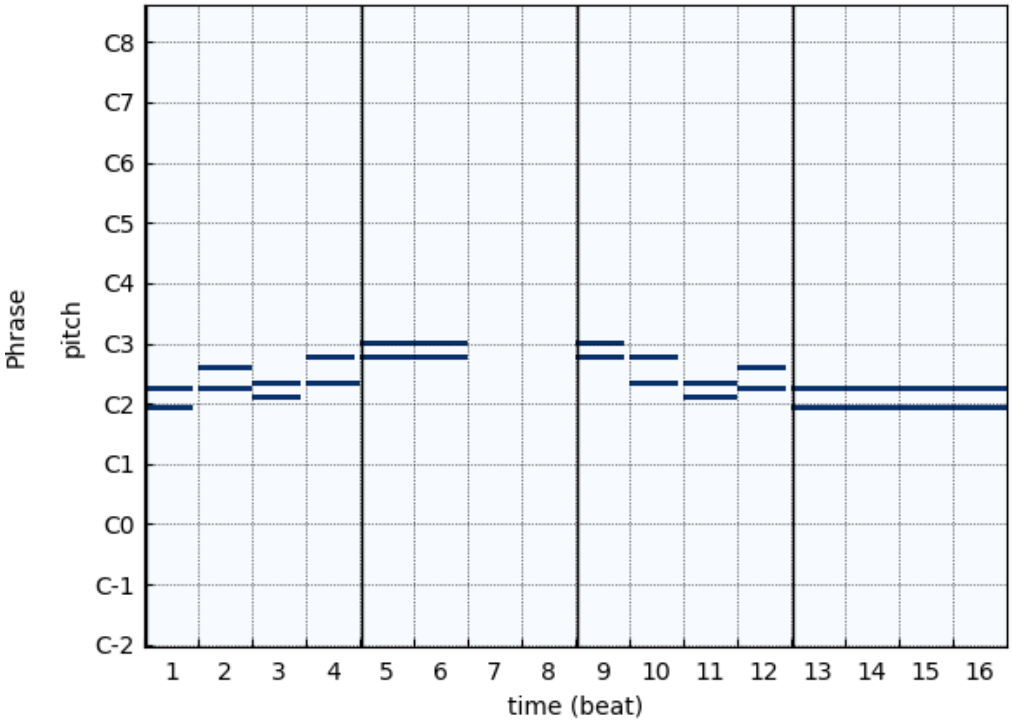


Figure 2.5.: Piano-roll representation.

2.2. Musical Instrument Digital Interface

In order to allow digital interfaces to communicate musical information, a protocol called Musical Instrument Digital Interface (MIDI) was created by Dave Smith and Ikutaro Kakehashi. They presented their creation at the 1983 January NAMM show in Anaheim, California (MIDI Association, 2015). MIDI works by sharing data between digital interfaces. The data being shared contain information about the music that is to be played, such as note pitch, velocity, pitch bend and so forth (de Oliveira and de Oliveira, 2017). A MIDI file can contain only a single track, several simultaneous tracks or multiple independent tracks.

2.3. Markov Chains

A Markov chain is a model that consists of random variables that satisfy the Markov condition, that the next state is solely dependent on the previous state (Norris, 2012).

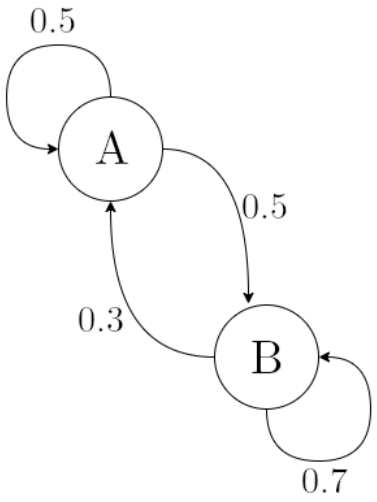


Figure 2.6.: Two-state Markov chain.

Figure 2.6 shows an example of a two-state Markov chain. The Markov chain works as follows: Say that we start in state A, and we are gonna move to the next state. The probability of moving to state B is 0.5, as indicated by the arrow in figure 2.6. The probability of moving to state A, i.e. staying in the same state, is also 0.5. These probabilities never change as long as we are in state A, no matter how many times in a row state A has been chosen. If we move to state B however, we get a new set of probabilities for the next state. Now the probability of the next state being A is 0.3, while staying in B is 0.7.

In order to use this model in a generative context, we perform what is called a *random walk*. This basically means that we use randomness and the probability distributions of the current state to choose the next state in the sequence. So even if the probability of the next state being A given the current state is B is 0.3, i.e the next state being B is more probable than next state being A, it is still possible for state A to be the next state because of randomness. Using random walk we are able to generate sequences like ABAB and AABA from the Markov chain in figure 2.6. By expanding the Markov chain to include, for example, the seven notes from the C major scale, we see how this can be used to generate simple musical sequences in the key of C.

2.4. Machine Learning

Machine learning uses training data in order to create a model that performs a specific task. The model is not explicitly programmed to handle the task, instead it learns by recognizing patterns from the training data. Machine learning has been used on a variety of tasks such as medical diagnosis and autonomous vehicles.

There are three main types of learning, each with their own algorithms (Russell and Norvig, 2009). *Supervised learning* is a field of machine learning that uses *labeled training data*. This means that the data the model is trained on has the input and the correct output of each training example, which the model can use to iteratively infer a function that generalizes well (Russell and Norvig, 2009). What this means is that the trained model is able to predict the output of previously unseen examples.

Unsupervised learning takes *unlabeled training data*, which means that the data has no correct output to base the inference of a function on. *Clustering* is the most common unsupervised learning task, which is when data is grouped together to detect clusters. These clusters can be used to detect patterns within each cluster, without there being explicit labels for each cluster, which can be used for classification or anomaly identification (Russell and Norvig, 2009). The third type of learning is *reinforcement learning*, which works by rewarding and punishing an agent based on its behaviour (Russell and Norvig, 2009). A chess agent, for example, will be rewarded for winning a game of chess. It is up to the agent however to determine which actions led to it being rewarded.

There is also a type of learning that is a hybrid between supervised and unsupervised learning called *semi-supervised learning*, where only some of the training data is labeled (Russell and Norvig, 2009). There may also be falsehoods in the labeled data. Semi-supervised learning uses both the labeled and unlabeled data to fit its model.

This thesis will mainly focus on supervised and semi-supervised learning methods, as they are the most relevant and prevalent in the field of computational musical creativity.

2.4.1. Artificial Neural Networks

Artificial neural networks (ANN) are a way of computing a result, inspired by neurons found in the human brain (Russell and Norvig, 2009). Neurons work by taking an input, and if the input exceeds a threshold, it fires. A neural network consists of multiple neurons, called nodes, which are connected and together generate the output. The connections have weights which adjust as the learning process progresses. The layer with the nodes are called the *hidden layer*. Figure 2.7 shows the structure of an ANN with one hidden layer and four nodes.

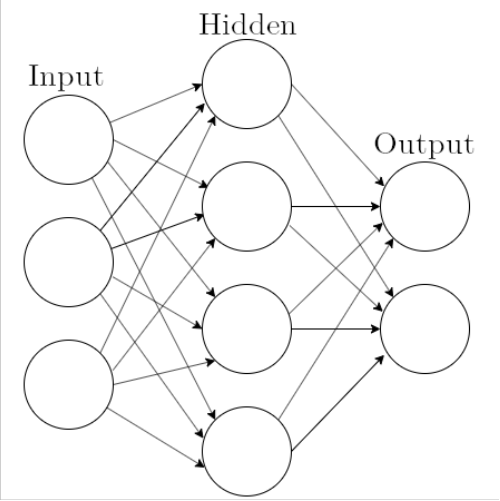


Figure 2.7.: Artificial Neural Network.

ANNs are typically used in supervised learning. This means that the training data is labeled, i.e. it has a set of examples and their corresponding label. By using the examples in the data set, the ANN predicts the label and based on the label given to the set of examples in the training data, the ANN adjusts its weights to increase the accuracy of predicting the labels of the examples.

2.4.2. Activation function

An activation function is the function each node uses to generate its output. The activation function can be either a hard or logistic function (Russell and Norvig, 2009). The output b_h of a node is a sum of the input vector x_i from i to n, the weights for each connection $w_{i,j}$ and the activation function θ_h (Graves, 2012):

$$b_h = \theta_h \left(\sum_{i=1}^{n} x_i w_{i,j} \right)$$

Below are descriptions of some commonly used activation functions.

Tanh Tanh transforms the input to the neuron into the range -1 to 1 (Nwankpa et al., 2018). The function is defined as:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

An advantage of this activation function is that it is zero-centered, which helps in backpropagation. A disadvantage of tanh is that it can generate dead neurons, when the input is 0.

Sigmoid This function transforms the input into the range 0 to 1 (Nwankpa et al., 2018). The function is defined as:

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

Softmax The softmax function is performed on an input vector. The way it works is that it normalizes the input vector into a probability distribution, with values between 0 and 1, where the sum of all the vector elements equals 1 (Nwankpa et al., 2018).

$$softmax(x_i) = \frac{exp(x_1)}{\sum_j exp(x_j)}$$

Due to the output being a probability distribution, softmax does not work with multi-label classification problems.

ReLU In deep neural networks, the rectified linear activation function is a commonly used activation function. Units that uses this function are called *rectified linear units* (ReLU). It is a fairly simple function, where input values below zero are returned as zero, while input values above zero gets returned as the input value (Nwankpa et al., 2018).

$$ReLU(x) = max(0, x)$$

2.4.3. Recurrent Neural Networks

Recurrent neural networks (RNN) is a class of ANN that allows the network to use previous events to determine the prediction for the next event, i.e. it remembers (Graves, 2012). This is done by using loops in the architecture of the network, which allows the information to persist in the network.

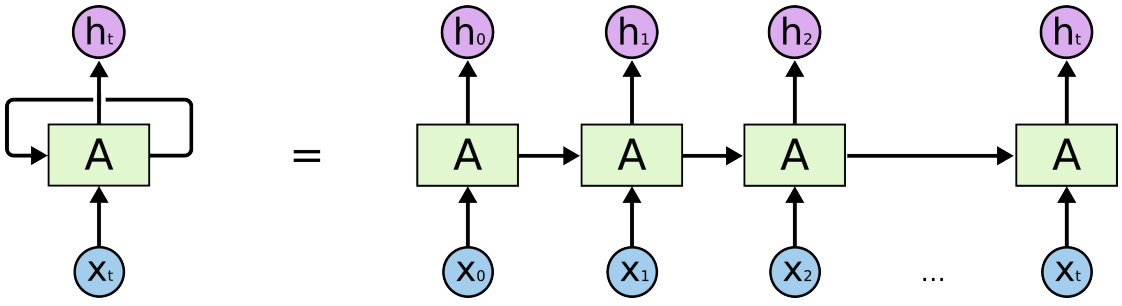


Figure 2.8.: Recurrent Neural Network[3].

[3]Reprinted from Colah's Blog, by Christopher Olah with permission from the author. Retrieved from: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Figure 2.8 shows the general architecture of an RNN. x_t is the input, A is the hidden state, and h_t is the output. The rolled RNN is how the concept of an RNN is visualized, with information looping back to the hidden layer and persisting previous events. However, in order to actually be able to implement this concept we have to "unroll" the loop, and create several identical networks that feed information to each other. In figure 2.8, the network with input value x_0 will be the first network to make a calculation, and it will feed its hidden state into the next network which takes input x_1 and so forth.

Long Short-Term Memory (LSTM) is a type of RNN that tries to fix the *vanishing gradient problem* (Hochreiter, 1998). The problem is that the sensitivity to the input for the nodes in an RNN decays as time passes. This means that the network forgets inputs over time. LSTMs deals with this by having a set of recurrently connected subnets called memory blocks, which contain self-connected memory cells (Graves, 2012). Also, the cells have three multiplicative units, input, output and forget gates, which makes it possible for the cells to read, write and reset themselves. These units make it possible for the memory cells to preserve information for a long time. The architecture of an LSTM cell is shown in figure 2.9.

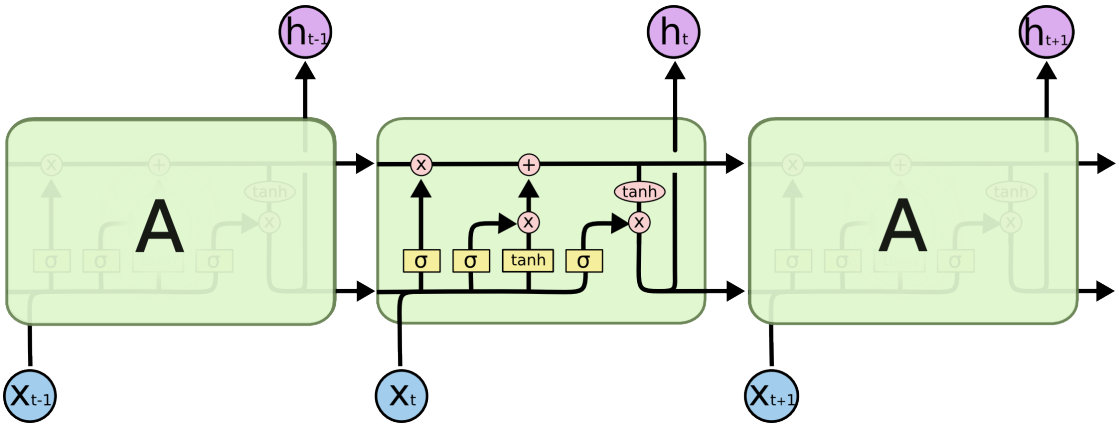


Figure 2.9.: LSTM cell with three multiplicative gates[4].

The sigmoid function is used in the "forget gate layer" in LSTMs in order to adjust the information in the cell state C_{t-1} that is passed through. The hidden state h_{t-1} and the input data x_t is used to do these calculations. If the result of the sigmoid function is 0, it means the current information in the cell state should be completely forgotten, while 1 means that it should be kept (Olah, 2015).

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

[4]Reprinted from Colah's Blog, by Christopher Olah with permission from the author. Retrieved from: `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`

The next step is the input gate, which is a sigmoid layer that produces the vector i_t that indicates which values should be updated. After this, the next layer is a tanh layer which is used to calculate a vector of new candidate values, $\tilde{C}_t$. The output of the input gate and the new candidate vector is combined in order to update the state.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

After this, the changes is applied to the cell state C_{t-1}, which gives the new cell state C_t.

$$C_t = f_t * C_{t-1} + i_t + \tilde{C}_t$$

The last step is calculating the output. The output is obtained by applying a sigmoid function to the cell state. After this, a tanh function is applied to the cell state. The results of these two functions is multiplied together, which results in only the relevant information being the output (Olah, 2015).

$$o_t = \sigma(W_o[h_{t-1}], x_t[+b_o])$$
$$h_t = o_t * tanh(C_t)$$

Sequence-to-sequence An encoder-decoder architecture that takes an input sequence and produces an output sequence based on the whole of the sequence and not just individual values in it (Sutskever et al., 2014). The encoder and decoder are both RNNs. The encoder takes the input sequence and outputs it's hidden state as a feature vector. The decoder then takes the feature vector, and decodes it producing an output sequence. An example of the usage of seq2seq is machine translation, where the translation of a sentence is not based on just each word in the sentence, but the sentence as a whole.

2.4.4. Convolutional Neural Networks

Convolutional neural networks (CNN) are a class of neural networks that is widely used today in the field of image recognition. CNNs are neural networks that contains at least one convolution layer, which is a layer that uses a linear operation called convolution instead of regular matrix multiplication (Goodfellow et al., 2016). Convolution is an operation on two functions of real-valued argument. The product of this operation produces a third function. The first function is called the input, while the second is called the *kernel*. A kernel is a small multi-dimensional matrix whose values determine the effects of the convolution. By altering these values it is possible to achieve effects such as

edge detection, blur and sharpening. By doing the convolution, the effect is applied to the image. Figure 2.10 shows an example of a convolution using a kernel of size 2x2. For the sake of simplicity, the kernel matrix is $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, i.e. no effect is applied. The kernel will move 2 positions for each operation.

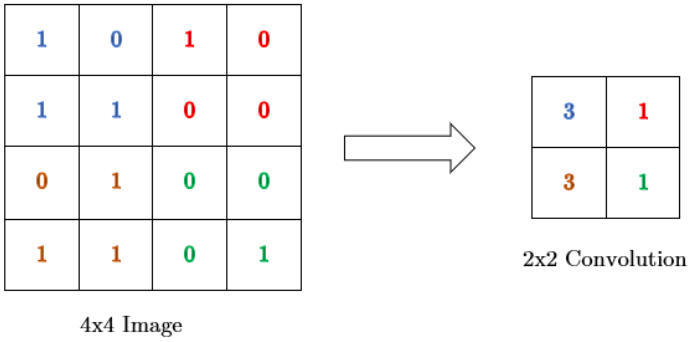


Figure 2.10.: Convolution using a 2x2 kernel on a 4x4 image.

The convolution starts with the 2x2 kernel being applied to the blue values in the 4x4 image. The kernel = $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, so we get a summation of the blue values in the image and the kernel values:

$$(1 \times 1) + (1 \times 0) + (1 \times 1) + (1 \times 1) = 3$$

The sum of this operation gives us 3 in the first cell of the 2x2 convolution.

Next the kernel moves 2 steps to the red values in the 4x4 image. We do the same summation:

$$(1 \times 1) + (1 \times 0) + (1 \times 0) + (1 \times 0) = 1$$

The summation gives us 1 in the second cell in the 2x2 convolution.

The kernel then moves to the brown values and applies the summation, and finally to the green values. Convolution on the 4x4 image in figure 2.10 using the kernel $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$, gives us the 2x2 convolution $\begin{bmatrix} 3 & 1 \\ 3 & 1 \end{bmatrix}$.

2.4.5. Generative Adversarial Network

In 2014, Generative Adversarial Networks (GAN) were presented by Goodfellow et al. (2014). The idea behind GAN is that two neural networks are adversaries, where the generative model G captures the data distribution and the discriminative model D tries to find out whether the sample came from the training data or G (Goodfellow et al., 2014). By doing this G tries to fool D, and D tries to expose G. The whole system is trained using backpropagation, which aims to improve the accuracy of each network. GANs are generative, meaning they do not try to reproduce data. Instead, they try to create new data indistinguishable from the training data.

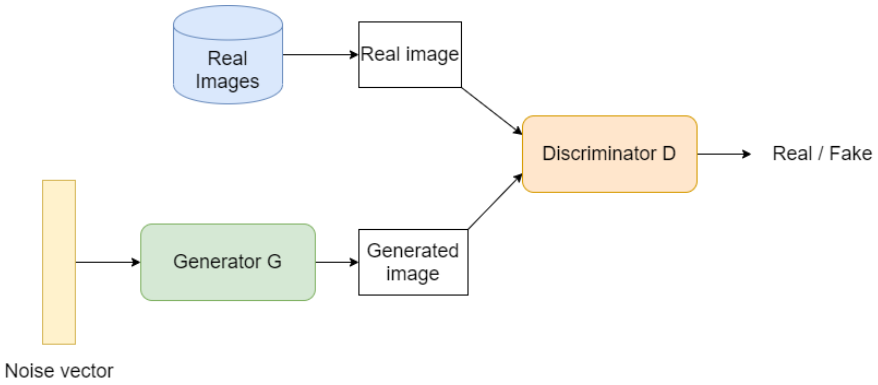


Figure 2.11.: GAN architecture.

This method of training D and G is described by Goodfellow et al. (2014) as a two-player minimax with the following value function $V(G, D)$:

$$\min_G \max_D V(D,G) = \min_G \max_D (E_{x \sim P_{data(x)}}[\, logD(x)] + E_{z \sim P_z(z)}[log(1 - D(G(z)))]\,)$$

GANs have been applied in a variety of fields, such as image generation, image upscaling and music generation.

Vanishing Gradient is a problem not exclusive to GANs, but one which GANs are prone to. It occurs when the gradients that are used to adjust the weights vanishes, which means that the weights are not updated. In GANs, this is caused by the discriminator becoming too strong, so that the gradient for updating the generator vanishes.

Mode Collapse is a problem that is quite common when training GANs. Goodfellow (2016) states that mode collapse is when the generator learns to map several different input values to the same output value. In other words, the same output is produced no matter what the input is. This is a problem that is widely researched, and some techniques have been proposed to solve the problem.

Freezing is a technique that is used in order to ensure that the generator or the discriminator does not become too strong. It means that if the loss of the generator becomes

much larger than the loss of the discriminator, the weights of the discriminator is frozen, allowing the generator to train on the training data and update its weights. When the loss of the generator is again within margin, the discriminator is unfrozen and can continue adjusting its weights. Freezing can also be used to do pretraining for the generator and the discriminator.

One-sided label smoothing replaces the 0 and 1 targets for a classifier with what is called smoothed values (Salimans et al., 2016). An example of smoothed values is 0.1 and 0.9. Label smoothing is meant to ensure that the classifier does not become too confident in its prediction. With one-sided label smoothing, only one target is converted into a smoothed value, for example 1 is converted to 0.9.

Feature matching is a technique which prevents the generator from overfitting to the current discriminator (Salimans et al., 2016). This is done by giving the generator a new objective. The new objective requires the generator to generate data that matches the statistics of the real data, instead of matching the real data itself. The discriminator is used to specify these statistics. These statistics are the expected value of the features in an intermediate layer in the discriminator. The discriminator is trained as usual, with it trying to separate the real samples from the ones generated by the generator. The objective function of the generator can be defined as, with f(x) being the activations in the intermediate layer in the discriminator:

$$||E_{x \sim p_{data}} f(x) - E_{z \sim p_z(z)} f(G(z))||_2^2$$

2.5. Evolutionary Algorithms

Evolutionary algorithms (EA) takes its cues from biology, specifically evolution (Bäck, 1996). The algorithm works in an iterative manner, where each iteration is called a generation. The first thing that happens is the generation of the initial population. After this, each individual's fitness is evaluated based on a fitness function. The fitness function is an evaluation metric that takes a generated solution to the problem as input and gives an output of how good of a "fit" the solution is. The next step is an iterative process called *parent selection*, that consists of selecting the individuals that will be used to produce the next generation. Usually, the individuals with the best fitness are selected for reproduction. The new individuals are bred through operations called *crossover* and *mutation*.

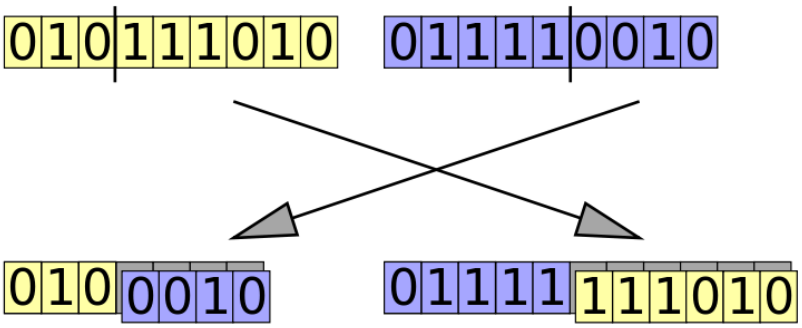


Figure 2.12.: Crossover[5].

Figure 2.12 shows an illustration of how a simple crossover operator works. We have two parents with different bit patterns. The first child inherits the first three bits from the first parents. Then we have the crossover, where the rest of the first child's bits is inherited from the second parent. This happens inversely for the second child. There are several different crossover operators which can be used.

Mutation is when one or several of the child's bits is flipped to be different from the inherited bits from the parent. An example of mutation is if the bit in the last position of the first child is flipped from 0 to 1. Mutation is dependent on a mutation probability p_m, where each bit has the mutation probability p_m to be flipped (Bäck, 1996).

The iterative process is supposed to continue until the solution has converged, i.e. the fitness function has reached the global maximum or minimum. However, this is often not the case and there has to be some termination clause. One common clause is to check the highest fitness of the population, and compare it over several generations. If the highest fitness does not change, the solution with the highest fitness is chosen as the final solution and the algorithm is terminated.

[5]Crossover by Josep Panadero is licensed under CC BY 3.0, link: `https://commons.wikimedia.org/wiki/File:Computational.science.Genetic.algorithm.Crossover.Cut.and.Splice.svg`

3. Related Work

The first music composition generated by a computer was the Illiac Suite for String Quartet in 1957 (Hiller and Isaacson, 1959). It demonstrated that a computer was able to generate something that could be considered music. After this initial success, the field of computer-generated music expanded, with people approaching computer-generated music from a variety of angles.

In the infancy of computational musical creativity a lot of research was done on using Markov chains to generate music. The fourth experiment of the Illiac Suite used this technique in order to generate the music (Hiller and Isaacson, 1959). Moorer (1972) shows the usage of Markov chains in generating short melodies. This research found that the Markov chains was able to capture some melodic characteristics.

This chapter presents a general overview of the historic work done in the field of automatic accompaniment generation, and relevant work done in the broader field of computational musical creativity. It also gives an overview of the state-of-the-art technologies.

3.1. Computer Accompaniment

Computer accompaniment has been researched a lot through the years. One prevalent researcher who has done extensive reasearch in the field is Roger Dannenberg. Dannenberg (1985) presented algorithms that could be used to accompany a soloist in real-time. The algorithms tries to synchronize a predefined score to what is currently being played by the soloist.

In 1987, Dannenberg and Mont-Reynaud presented a model that tries to follow a jazz solo by playing chords that fits the current melody being played. The performance of the model proved to be encouraging but not high. In their conclusion, Dannenberg and Mont-Reynaud (1987) suggests that in the future, intelligent programs will enrich the relationship between computers and humans in music.

Dannenberg and Xia (2017) aimed in their paper to bridge automatic accompaniment and computer-generated improvisation. In other words, they sought to make the computer play the notes in the score, while also adapting to what the soloist is playing. By doing this, the automatic accompaniment learns to accompany a soloist by adding, modifying and removing pitches and rhythms. The computer constructs its accompaniment based on the chords that are in the score. The model Dannenberg and Xia created uses nearest-neighbor search to produce its output. The primary feature that determines the output is what they call onset density, which is a measure of the number of score onsets. The secondary feature is chord thickness, i.e. the average number of notes in each chord.

The subjective evaluation by Dannenberg and Xia (2017) was that their proposed model performed very well, and they concluded that they had been successful in creating a virtual accompanist with basic improvisation skills.

Dannenberg's research focuses in broad terms on accompanying improvisation using a predefined score. Generating an accompaniment based on a melody is a bit different. First of all the melody is fixed, i.e. the model has access to the whole sequence of notes and is able to base its output at time t on the notes at time $t + 1$ of the melody as well as all the notes prior to t. In an improvisation context, an accompanist only has information about the sequence $[0, 1...t_{-1}]$ and can only base the choice of note at time t on this information. Also, when generating an accompaniment based on a melody the model does not have a score to follow, but rather has to pick its output based on the training data it has learnt from.

3.2. Artificial Neural Networks

With the increase of available computational power and data, the prevalence of Markov chains have diminished, and artificial neural networks (ANN) have become widely used. Some of the networks most commonly used are recurrent neural networks (RNN). This is because of their ability to remember previous runs of the network.

3.2.1. CONCERT

CONCERT was a system, developed by Michael Mozer in 1994, that utilized RNNs to generate new melodies (Mozer, 1994). The system performed well on simple, structured artificial sequences but failed to generate natural music. Mozer (1994) writes that the generated pieces were not musically coherent. He argues that this makes sense, since CONCERT learns the structure of music through a note-by-note analysis. Mozer finds this analogous to learning English by doing a letter-by-letter analysis of a text.

3.2.2. Automatic Accompaniment Generation

ANNs have also been applied to the field of automatic accompaniment generation. Qi and Zou (2016) proposed a model that adds the accompanying chords and rhythm for a melody by using recurrent neural networks. The choice of RNNs comes down to their ability to effectively capture sequential data (Karpathy, 2015). Their implementation is what is called a sequence-to-sequence network, which means that the network takes an input sequence and produces an output sequence based on the whole of the input sequence and not just individual values of the input sequence. The findings of this research was that the generated accompaniment was relatively simple, with a lot of repetition. The rhythm section turned out to be able to accurately capture the rhythm of the melody.

3.3. Evolutionary Algorithms

Evolutionary algorithms have also been used in computational musical creativity. One of the more well-known examples of this is GenJam, which uses a genetic algorithm-based model in order to generate jazz solos (Biles, 1994). GenJam was developed by John Biles, a jazz musician and computer scientist. The results of the system were obtained through real-time feedback, which helped shape the performance of the system through fitness values given to different measures and phrases. These are then used to generate a new generation, which is obtained through crossover and mutation. The produced artefacts are available online, and are subjectively judged to be musical. Some of the produced GenJam artefacts have even been released as songs on an album in 1996[1].

3.4. State-of-the-Art Technologies

This section presents the current state-of-the-art technologies and techniques that have been applied to the field of computational musical creativity.

3.4.1. Magenta

Magenta is an open-source research project developed by researchers and engineers at Google Brain. It aims to explore using machine learning as a tool in creative processes such as music, art and poetry[2]. With regards to music, it is able to create music just by the user inputting the desired length, key and beats per minutes of the generated song. This however produces results with a lot of silence and random notes without any musical motifs. Magenta is much better at generating music if it receives the start of a music piece on which it can build on. If we, for example, send in the first three notes of the song "Mary Had a Little Lamb" to the model, we get a generated song that is not the same as the piece it is based on, but the model still produces a music piece that is subjectively judged to be musical.

The architecture of Magenta is built using TensorFlow, Google's open-source machine learning platform. Magenta has a wide variety of pre-trained models that are able to generate music. The model that is to be used can be specified by the user. Some of the models implemented uses algorithms such as RNNs, generative adversarial training and long short-term memory.

3.4.2. AIVA

AIVA, which stands for Artificial Intelligence Virtual Artist, is an AI that composes original musical pieces. AIVA is trained on the music of composers such as Mozart, Beethoven and Bach[3]. Since AIVA is a commercial product, the details behind the

[1] https://www.last.fm/music/Al+Biles+Virtual+Quintet
[2] https://magenta.tensorflow.org/
[3] https://www.aiva.ai/about

architecture are not shared other than that deep learning and reinforcement learning was used as the machine learning techniques in order to achieve its results.

3.4.3. MuseGAN

MuseGAN is a project by Dong et al. which aims to generate realistic music. They modeled multi-track, polyphonic music using multiple-track piano-roll representation (Dong et al., 2018a). This allowed them to implement a GAN that uses convolutional neural networks. Their implementation consists of three models, each with their own strengths and weaknesses. The first model is what they define as the *jamming model*, where multiple generators work independently in generating music. Each generator has a discriminator that critiques the output. The second model is what they call the *composer model*. This model is a single generator that generates all the tracks, and a single discriminator that critiques the output. The last model is the *hybrid model*, where there are multiple generators and a single discriminator (Dong et al., 2018a).

MuseGAN has two different methods of generating music. The first generation mode generates music from scratch, without any kind of user input. The second method is what is called track-conditional generation. This method takes an existing piece of music and generates the accompanying parts based on it. The models were evaluated through two studies, namely the objective evaluation and the user study. In the objective evaluation, each model and method is given four intra-track and one inter-track metric:

- **EB:** Ratio of empty bars.
- **UPC:** Number of used pitch classes per bar.
- **QN:** Ratio of notes longer than three time steps (32th note). This is what they call a "qualified" note.
- **DP:** Measure of drum pattern. Ratio of notes 8- or 16-beat patterns.
- **TD:** Harmonic relations between the tracks.

In the four intra-track metrics, the generated tracks are evaluated against values of the training data. For the inter-track metric TD, a smaller value is considered better. The results by Dong et al. finds that the jamming model tends to perform the best in the intra-track metrics, but is the worst at the inter-track metric. This goes for both the from scratch and track-conditional generation. There is no clearly superior model between the composer and hybrid model in the intra-track or inter-track metrics. The TD scores of both the composer and hybrid model are very similar across combinations of tracks.

The user study was a listening study, where 144 respondents gave scores to some generated music in the terms of how harmonious, rhythmic, musically structured and coherent they thought the generated tracks were. They also gave an overall rating. All these ratings were given on a 5-point scale. Of the 144 respondents, 44 were considered

"pro user" with regards to their musical background. These 44 were separated into a separate group called pro, and the remaining 100 were the non-pro group.

The user study found that the overall best performing model was the hybrid model when generating music from scratch. It got an overall rating of 3.16 from non-pros, and a 2.93 from pros. Of the track-conditional models, non-pros found the jamming model to be the best, with an overall rating of 3.06. Pros believed that the hybrid model performed best at track-conditional generation, giving it an overall rating of 2.70.

Dong et al. (2018a) concludes that the proposed models can start to learn something about music, though their generated artefacts are found to be musically and aesthetically lacking compared to music composed by human musicians.

3.4.4. C-RNN-GAN

Continous RNN-GAN (C-RNN-GAN) is a recurrent neural network architecture trained using adversarial training (Mogren, 2016). RNNs were chosen because of their abilities to model sequences of data. This makes it a good fit for generating music, since music has a sequential structure. Mogren (2016) took inspiration from the MIDI-format in order to represent notes, and used the four scalars *tone length*, *frequency*, *intensity* and time spent since the previous tone to model each individual note. The data set he used were a total of 3697 midi files from 160 different classical composers. Mogren (2016) evaluated C-RNN-GAN using four metrics on the generated output. These metrics were:

- **Polyphony:** How often are two tones played simultaneously.
- **Scale consistency:** How many tones were part of a standard scale.
- **Repetitions:** How much small subsequences repeated.
- **Tone span:** The number of semitones between the lowest and highest tone in a sample.

The evaluation was done using a Python-script. Two different variations of the architecture were tested, one using *feature matching* and one using feature matching as well as three tone outputs per LSTM cell. Feature matching is a technique used to encourage greater variance by the generator, so that it does not overfit to the discriminator. These two variations were evaluated against a baseline RNN model, and the training data itself.

Mogren (2016) concludes that more experimentation is needed, but the results of the system are promising. C-RNN-GAN showed more resemblance to the training data than the baseline model did. The two different variations performed similarly in most of the metrics, but allowing the LSTM cells to output three tones greatly increased the polyphony metric. By human judgment however, the generated music was deemed to not be comparable to the training data (Mogren, 2016).

4. Architecture

This chapter gives an overview of the implemented GAN architecture. In order to be able to train this implementation, training data is required. This is described in the first section of this chapter.

4.1. Training data

The training data used is a 5-track piano-roll dataset[1]. This training data is a processed dataset of the Lakh Pianoroll Dataset, which was used in MuseGAN (Dong et al., 2018a).

The training data consists of 34,126 phrases, with each phrase being 8 bars. Each bar in the 8-bar phrase consists of 48 time steps, where each time step is a piano-roll representation of the the pitches played at that specific time. This means that each 8-bar phrase consists of a total of 384 time steps. Each time step has a tonal range of 84. This is represented as an array consisting of 84 boolean value, representing the span of 84 pitches in the span C1 - B7. If a pitch is played, the boolean value in the index representing the pitch is *True*, while non-played pitches are *False*. This means that pauses are represented as an array of 84 boolean values who are all False. Table 4.1 shows the array of a C major chord consisting of the pitches C3-E3-G3. The rows represent the octaves, with row 1 being octave 1, and the columns represent the pitches, starting with C.

	C	C♯	D	D♯	E	F	F♯	G	G♯	A	A♯	B
1	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False
3	*True*	False	False	False	*True*	False	False	*True*	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False
5	False	False	False	False	False	False	False	False	False	False	False	False
6	False	False	False	False	False	False	False	False	False	False	False	False
7	False	False	False	False	False	False	False	False	False	False	False	False

Table 4.1.: Example of a C major chord (C3-E3-G3) in a time step array consisting of 84 boolean values.

[1]Dataset available from here, marked as version 2: `https://github.com/wayne391/symbolic-musical-datasets/tree/master/5-track-pianoroll`

For each of the phrases in the training data there are five tracks, representing each part played by five different instruments. The five instruments are *Drums*, *Bass*, *Guitar*, *Strings* and *Piano*. The training data is stored in a .npy-file, which is a file-type used by *NumPy* for storing data[2]. NumPy was also used in order to manipulate the training data. The total size of the data is 5.12 GB. In order to use this data as input in the model, the truth values of each time step is converted into float, with True being 1.0 and False being 0.0. This allows the system to use the training data in its calculations. Figure 4.1 shows the first 4 bars of a guitar part and the corresponding bass part for a sample phrase from the training data.

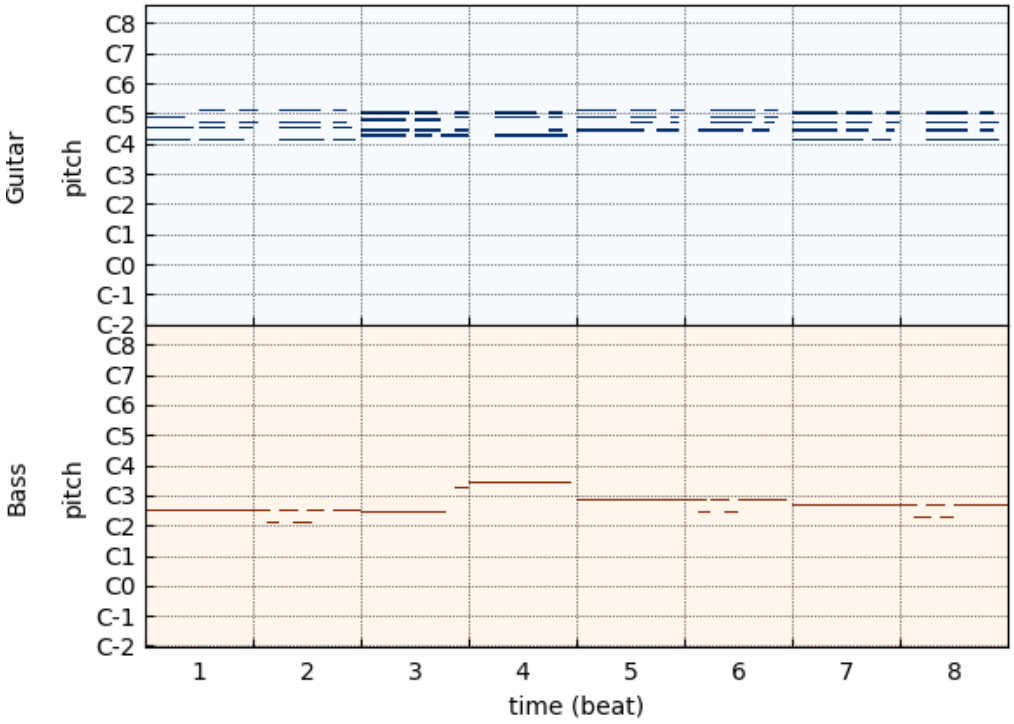


Figure 4.1.: Sample phrase from the training data for a guitar and bass part.

The system uses the guitar part as the training data, and the goal of the system is to generate a fitting bass accompaniment. Generating bass accompaniments was chosen because of the complexity involved with generating chords. Chords consists of multiple notes being played at the same time. Guitar and piano are usually the instruments playing chords in songs. The bass for the most part plays single notes. By generating bass accompaniments, the system usually needs to output a single label. It is possible for

[2]About .npy: `https://numpy.org/doc/stable/numpy-ref.pdf`

the system to generate chords, by for example outputting the three notes with the highest probability. This however causes the model to only play chords. A possible solution to this is to have a confidence threshold, making it so that notes that are within for example 0.1 of the probability of the label with the highest probability are also played. This was however not implemented in this architecture.

Tables 4.2 and 4.3 presents some statistics for the training data. In order to obtain these statistics, the Python package *pypianoroll* was used (Dong et al., 2018b). These statistics will later be used when the evaluating the generated accompaniments by the system from the different experiments.

	Pauses	Chords	Notes	Uniques	Durations	Spans
Max	363	384	384	144	384	56
Min	0	0	0	1	1	0
Average	75.3	217.6	91.1	23.9	8.8	14.8

Table 4.2.: Statistics for the guitar phrases in the training data.

	Pauses	Chords	Notes	Uniques	Durations	Spans
Max	360	384	384	57	384	43
Min	0	0	0	1	1	0
Average	66.3	9.0	308.6	10.7	12.2	12.8

Table 4.3.: Statistics for the bass phrases in the training data.

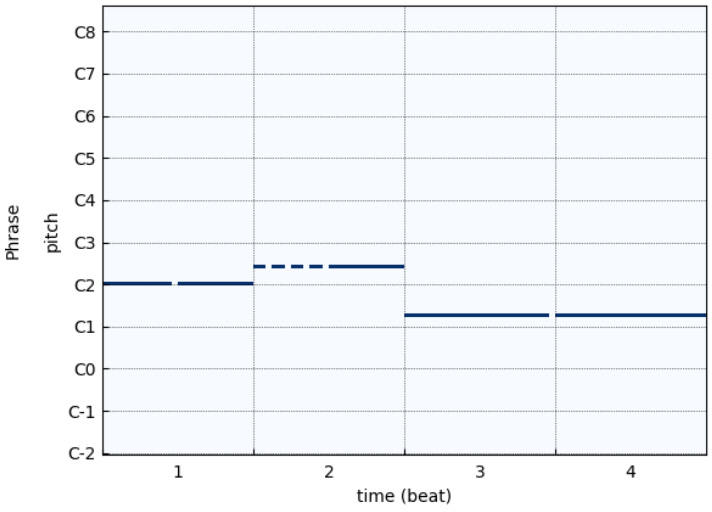


Figure 4.2.: Example of four unique events.

The two tables contain the statistics for the bass and guitar parts in the training data. Pauses is the total number of time steps that are pauses. One important thing to note here is that pypianoroll inserts a pause time step between note changes, and chord changes. Chords is how many time steps two or more notes are played at the same time. Notes are how many time steps a single note is played. Uniques are how many unique events are in the phrase. For example, a phrase such as figure 4.2 shows three unique notes being played, with a fourth unique event being the pauses. Therefore, the uniques score of figure 4.2 is 4. Durations is how many time steps notes, or chords, are held. Spans refers to the span between the lowest and highest note in the phrases. Spans is counted using semitones. The tables show that the phrases in the training data is very varied. For example, the amount of time steps that are pauses in a guitar phrase is on average 75.3. However, there are phrases where there are 0 time steps with pauses, and phrases where there are 363 time steps that are pauses. The statistics for the all instruments in the training data can be found in appendix A.

4.2. Generative Adversarial Network

The generative adversarial network (GAN) consists of two recurrent neural networks (RNN). The architecture of the networks are based on the C-RNN-GAN architecture by Mogren (2016)[3].

The GAN was implemented using *PyTorch 1.4*, which supports GPU-acceleration for training the system. To take advantage of this, a virtual machine with a GPU hosted on Google Cloud was used to train the GAN.

4.2.1. Generator

The generator is an RNN containing two LSTM layers. The initial layer is a linear layer, whose output is passed through a ReLU function. This is then used as the input to the LSTM layers. There is a dropout layer between the two LSTM layers, which has a dropout probability of 0.5. Lastly, there is another linear layer. The generator outputs an 8-bar accompaniment for a melody given as input. In order to generate the accompaniment, the input into the generator is a series of time steps of either 24 or 84 float values, with each value being either 0.0 or 1.0. The system allows for a reduced tonal range in order to reduce the overall span of the notes in the accompaniments. This is configured by a variable which indicates whether to reduce the tonal range of time steps or not. How many time steps that are used as input is defined in the configuration by the variable *time_steps*. If we want to generate output based on the previous six time steps, *time_steps* = 6. The experiments presented in chapter 5 tested different configurations for the training data, and how the generator produces its output. The final architecture allows for two different configurations for the generative process which are presented below.

[3]Source code available here: `https://github.com/olofmogren/c-rnn-gan`

In the first configuration C1, the input into the generator is the time steps g_{i-6} - g_{i-1} from the previously generated outputs, as well as the time steps t_{i-5} to t_i from the input phrase. If $i = 0$, the six time steps of previously generated outputs are set to be empty. For the input phrase, five empty time steps would be prepended, so that the input at time $i = 0$ would be five empty time steps and the first time step in the phrase. This approach means that each of the time steps in the generated 8-bar accompaniment is calculated based on a certain number of preceding time steps of the leading input phrase and the generated output. This is a technique called *sliding window*, where a window of a given size is moved through the data for each iteration, creating sub-samples of the data that is used to calculate the output.

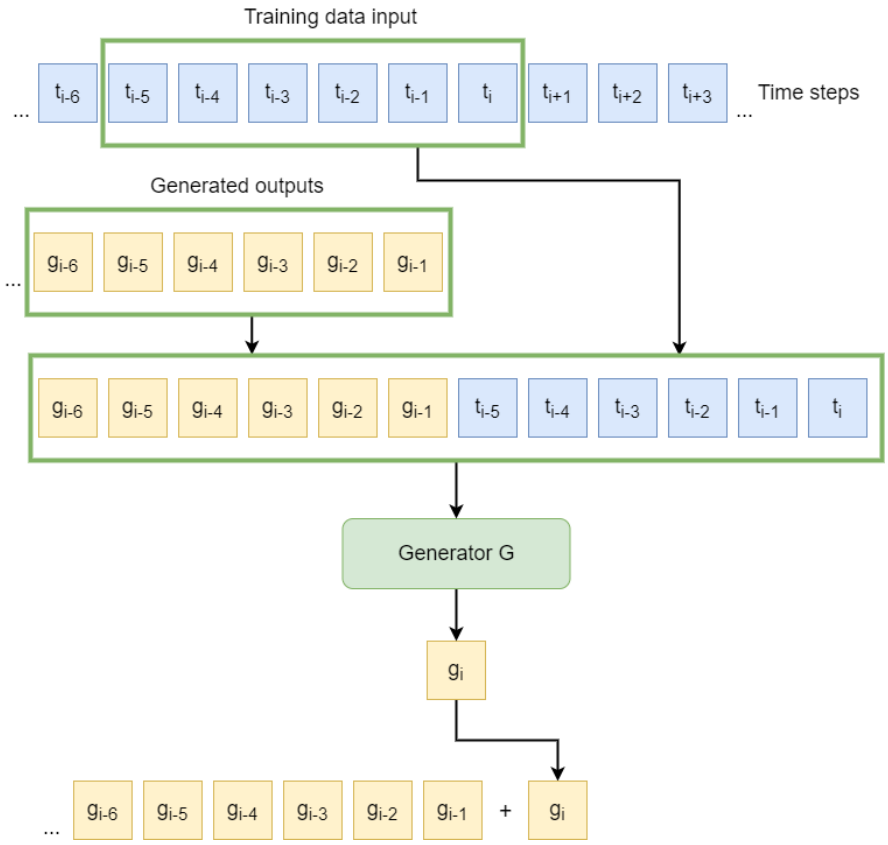


Figure 4.3.: C1 – Generation for a single time step with previous time steps being 6.

Figure 4.3 displays the process of generating of a single output time step using this configuration. In this example, the number of previous time steps for the input and previously generated is six. The input into the generator is a concatenation of the previous t_{i-5} to t_i time steps of the melody, and the g_{i-6} to g_{i-1} previously generated time steps. Therefore the total number of time steps used as input is 12. The input is then fed into the model, where each layer performs its calculations and outputs a single time step g_i. This time step is then appended to the time steps of the previous generations, and it will

be used in the generation of the next time step. The time steps g_{i-6} and t_{i-5} will be removed from the sliding window in the next iteration, when $i = i + 1$. This process is performed 384 times, resulting in the generation of 384 time steps, which is 8 bars.

The second configuration C2 for the generative process does not include the previously generated outputs in the input for the generation of the next time step. This means that the input to the generator is only the leading melody itself. This configuration was implemented due to the overall similarity displayed by the generated accompaniments in the first configuration. It was shown in experiment 6 that this configuration increases the note variations in the generated accompaniments. By using this configuration, the generative process of G can be described using figure 4.4

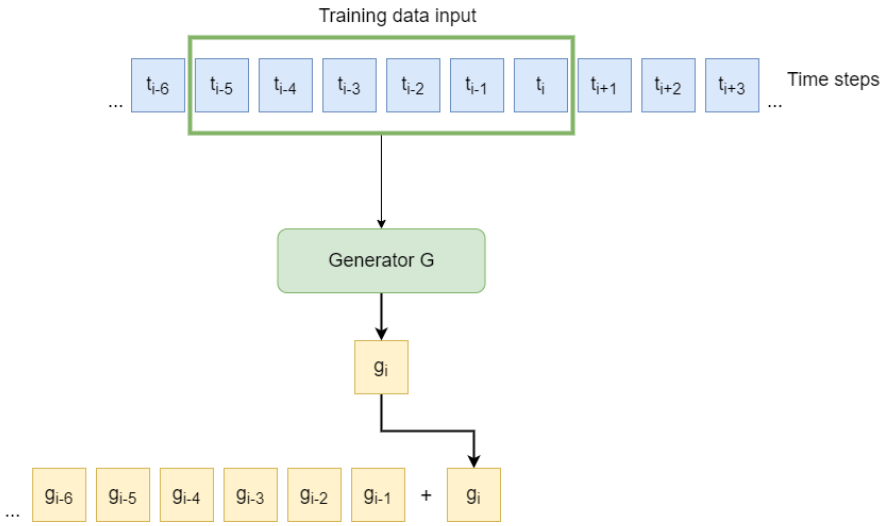


Figure 4.4.: C2 – Generation for a single time step with previous time steps being 6.

4.2.2. Discriminator

The discriminator, like the generator, also has a linear layer as the first and last layer. It also constains two LSTM layers. These two layers are bidirectional, meaning that they are able to get information from both the previous and future states. They also have a dropout probability of 0.5. The aim of the discriminator is to correctly distinguish between the "real" examples in the training data, and the "fake" examples generated by the generator. The discriminator takes only the generated phrase as input and judges it on its own merits. As with the generator, the discriminator looks at sequences of length equaling the specified *time_steps* in order to judge whether it is real or fake. Each time step in the input gets a score between 0 and 1, which gives a probability of the time step being real or fake. If the majority of time steps are given a score below 0.5, the accompaniment is judged to be fake. Inversely, if the majority of time steps have a score greater than 0.5, the accompaniment is judged to be real. The discriminator configuration is the same regardless of the configuration of the generator.

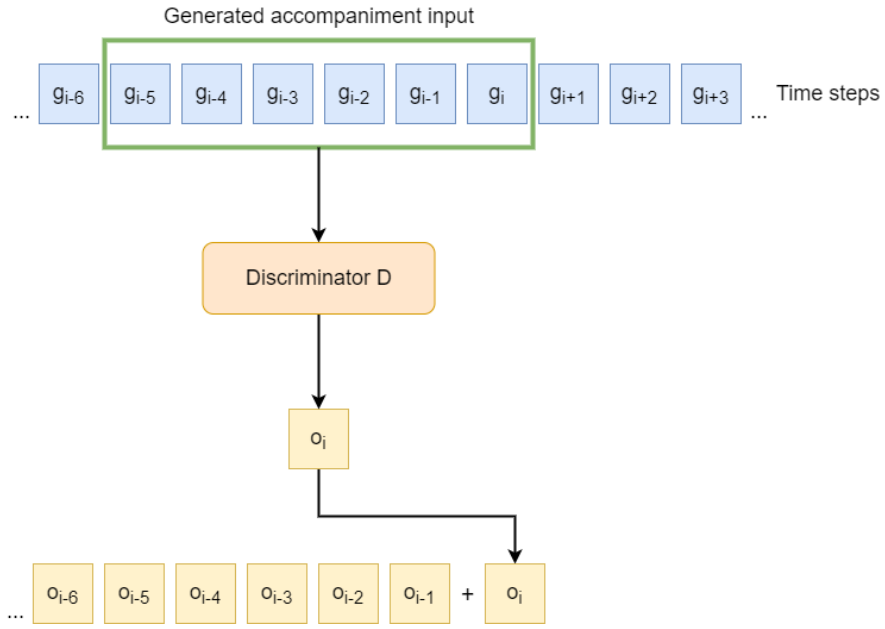


Figure 4.5.: Discriminator judging the generation with six previous time steps.

5. Experiments and Results

5.1. Initial Experiments

Before the final system was created, some initial models were proposed, implemented and tested. These proved to not be good solutions for the task of automatic accompaniment generation due to a variety of reasons. The first model used MIDI events as training data. When this approach proved to be unsuccessful, training data represented in piano-roll was used.

5.1.1. MIDI events

The initial suggested implementation for automatic accompaniment generation utilized the data found in MIDI files in order to train the model. The proposed training data for this was Lakh MIDI Dataset (Raffel, 2016). This however proved to be a problematic approach because of the way music is represented in MIDI format. In MIDI files, music is represented as MIDI events which contain information about the note or chord that is currently being played. This information includes pitch, velocity and duration, among other things. This means that each note or chord is a single MIDI event, no matter the duration of the note.

This way of storing information makes using MIDI format for accompaniment generation problematic, since there is not a one-to-one relation in what is being played by for example the bass and the guitar. If the bass is playing 16th notes, and the guitar is playing 8th notes, they would become out of sync since there would be double the amount of MIDI-events for the bass. This in turn causes the accompaniment generation to train on the wrong MIDI events, causing a time discrepancy problem as shown in figure 5.1.

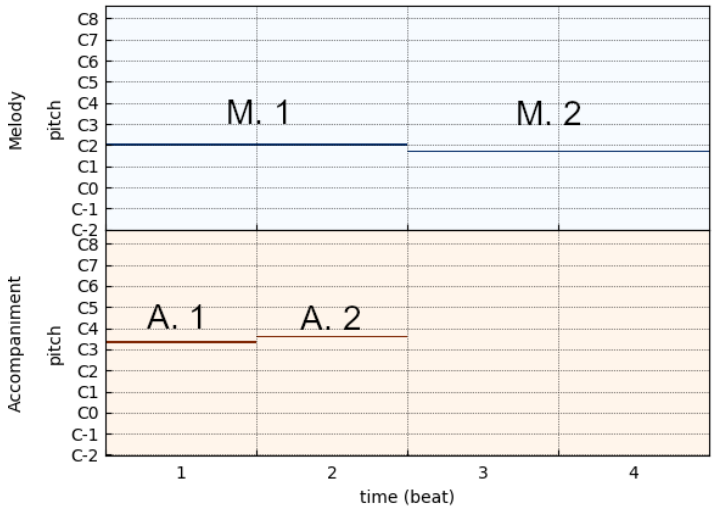


Figure 5.1.: Time discrepancy caused by different note durations.

Figure 5.1 shows the piano-roll representation for two MIDI events for the melody (M. 1 and M. 2), and two MIDI events for the accompaniment (A. 1 and A. 2). Using MIDI events as the training data, the accompaniment A. 1 would be based on the melody event M. 1. Likewise, A. 2 would be based on M. 2. But the combined durations of A. 1 and A. 2 equals the duration of the single event M. 1. Therefore, the melody and accompaniment would become out of sync, causing a time discrepancy in which the accompaniment trains on events that are not played at the same time.

This problem can also be present from the beginning, if the parts start at different times in the song. An example of this is given in figure 5.2.

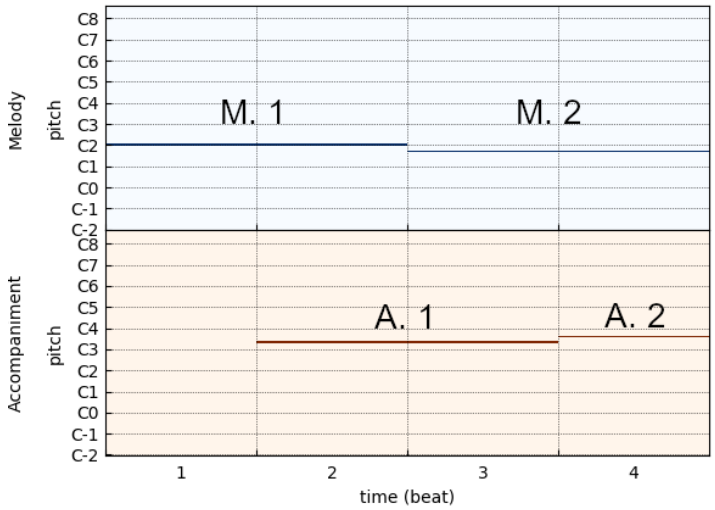


Figure 5.2.: Time discrepancy caused by different starts.

5.1.2. Unique piano-roll events

By using the data in piano-roll representation instead, the time discrepancy problem was solved. This approach took the 84 boolean values at each time step, and created unique events for each chord, note and pause. This was done by creating a string which concatenated the indices of each note played. So a C major chord consisting of C3-E3-G3 would be the string "24-28-31", with the indexing starting at zero. Pauses were given the string value "p". All these unique events were given a unique id in the corpus. The training data for the model then became a sequence of 384 unique ids of the events that were being played at each time step.

This approach is inspired by models in *Natural Language Processing* (NLP), where words are denoted using unique ids. This is because it is a lot easier to train models when the input is numerical instead of strings.

This approach did not work very well for automatic accompaniment generation. A problem with this approach is with the relationship between the unique events. In a musical sense, a C note and a C major chord are inherently related since a C major chord contains a C note. Therefore, it is logical that a generated C note is considered "more correct" than for example an F♯ note, which is not even in the C major scale. Using the approach inspired by NLP however, all events are considered equally unrelated. There may be a way of fixing this problem, which is also inspired by NLP. By using representations of chords and notes as vectors, akin to word vectors, relationships between chords and notes could be represented. This approach however would require calculating the vectors by analyzing relationships in vast amounts of musical data. Therefore, this approach was discarded in favor of another approach described in the succeeding sections.

5.2. Experiments

This section presents the experiments conducted that lead to the final architecture presented in chapter 4. A piano-roll representation of some samples of the generated accompaniments, as well as the real accompaniments, from these experiments can be found in appendix B. For all experiments conducted, a random sample of 100 phrases from the training data was selected. The batch size in all experiments was one.

5.2.1. Experiment 1: Multi-hot piano roll representation

In the training data, each time step is an array of 84 boolean values representing the pitches that are being played. This can be considered a multi-hot encoding of the currently played pitches. By using this representation as the input data, it is possible to generate both chords and notes, by treating the problem as a multi-label classification problem. This approach requires a loss function which is able to handle multi-label classfication. One such loss function is binary cross entropy. PyTorch has two built-in binary cross entropy functions, BCELoss and BCEWithLogitsLoss [1]. The difference is that BCEWithLogitsLoss performs a sigmoid function on the input. BCEWithLogitsLoss was chosen, because the output of the last linear layer may contain negative numbers. BCELoss does not work with negative numbers, so the data would need to be manipulated before it can be fed into BCELoss. One thing to note here is that in the training data, pauses are the absence of True values in a time step. This means that the loss function will not give a pause as the most probable label, because it will always give the label with the highest probability. One way of dealing with this is to use a confidence threshold, saying that if all probabilities are beneath a certain threshold, the estimated label is pause. This is an idea worth further exploration, though it has not been tested during these experiments.

The architecture for this experiment is inspired by C-RNN-GAN, with a few modifications to make it compatible with the piano-roll representation (Mogren, 2016). The generator G contains two LSTM layers, and linear layers as the first and last layer of the network. Before the output of the first linear layer is passed to the first LSTM layer, a ReLU function is applied to the output. Between the LSTM layers there is a dropout layer with a dropout probability of 0.5, per the default configuration of C-RNN-GAN. G receives the time step t_i in the input phrase, as well as the previously generated time step g_{i-1} by the model. Therefore the number of features that are used as input for G is 168, the two time step arrays of input phrase at t_i and generated output at g_{i-1}. This is done for all 384 time steps in the input phrase. When $i = 0$, the initial generated time step g_{i-1} consists of an empty array.

The discriminator D is a bidirectional RNN, consisting of a dropout layer with probability 0.5 as the first layer. Then there are two LSTM layers with a dropout layer between them with the same dropout probability. Lastly there is a linear layer

[1] More information about PyTorch's binary cross entropy functions: `https://pytorch.org/docs/master/nn.html#loss-functionsl`

which outputs a single value for each of the time steps in the input. D takes only the generated accompaniment as input, and judges each time step one after another.

The optimizer used in this architecture is Adam, which is the same optimizer as in C-RNN-GAN, set to a learning rate of 0.1[2]. The model was run for 50 epochs.

Table 5.1 is a summary of the architecture that was tested.

	Generator	**Discriminator**
Layers	2	2
Bidirectional	False	True
Input length	168	84
Output length	84	1
Hidden dim.	256	256
Optimizer	Adam	Adam
Loss	BCEWithLogitsLoss	BCEWithLogitsLoss
Learning rate	0.1	0.1

Table 5.1.: Summary of the architecture.

[2]More information about Adam: `https://pytorch.org/docs/stable/optim.html`

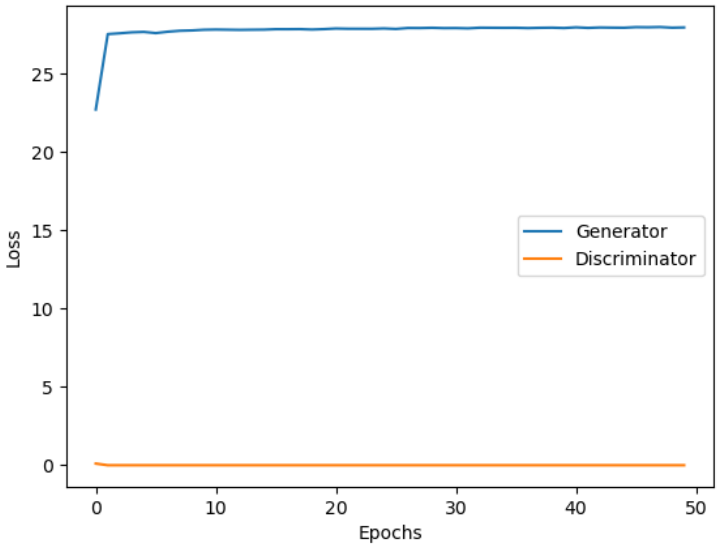


Figure 5.3.: E1 – Generator and discriminator loss.

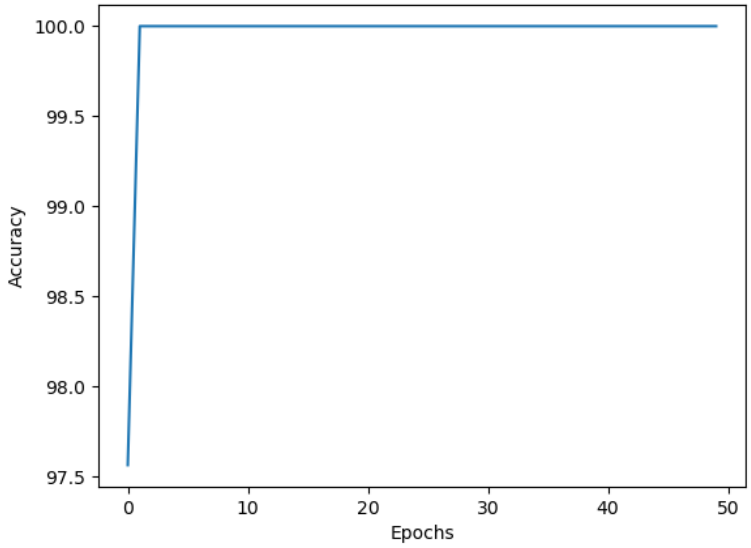


Figure 5.4.: E1 – Discriminator accuracy.

Using this configuraton, the loss for G is much greater than the loss of D, as seen in figure 5.3. The accuracy of D is 100% after a few epochs. With too strong of a discriminator, the resulting gradient cannot be used to improve G. This is what is called

a vanishing gradient problem.

5.2.2. Results of experiment 1

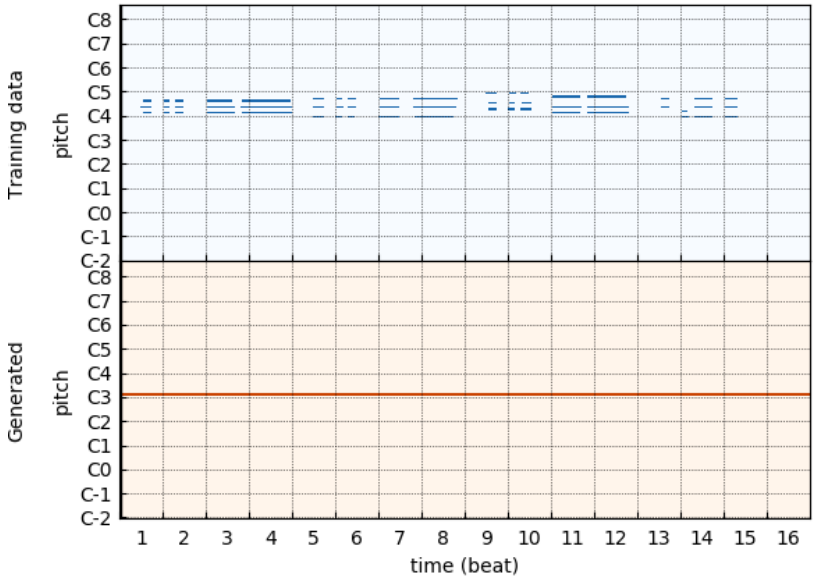


Figure 5.5.: E1 – Generated accompaniment and input melody.

When viewing the generated accompaniments, it is revealed that G outputs the same single note for all inputs. An example of this is displayed in figure 5.5. This points to the model also suffering from mode collapse, a common problem when training GANs (Goodfellow, 2016). Viewing the other accompaniments reveal that the system generates the same accompaniment for all melodies.

5.2.3. Experiment 2: Handling mode collapse and vanishing gradient

The C-RNN-GAN architecture implements multiple techniques in order to combat the problems of vanishing gradient and mode collapse. One of them is feature matching, where the output of the LSTM layer in D is used as the objective function of G. This was also implemented for this experiment.

Likewise, freezing was implemented. By freezing the weights of either D or G, the model ensures that the loss of either one does not become too large compared to the other. Freezing is also used in order for G and D to pretrain. The initial pretraining was set to six epochs for both G and D. Also, if D becomes too strong, it will be frozen until it is within a reasonable accuracy. This accuracy threshold was set to 95% accuracy. The pretraining starts by freezing G for six epochs, and updating D based on the output of

G. Then, after six epochs, G is unfrozen and D is frozen. G is then updated based on the labeling by D for six epochs.

Another technique that was implemented is one-sided label smoothing, which ensures that D does not become overconfident. This was done by setting the values of the truth vector to be 0.9 instead of 1.0. This is a technique suggested by Salimans et al. (2016) to help combat mode collapse.

The maximum size of the gradient was also limited to a maximum of 5.0, in order to try and prevent the discriminator from becoming too strong too fast, thus combating the vanishing gradient.

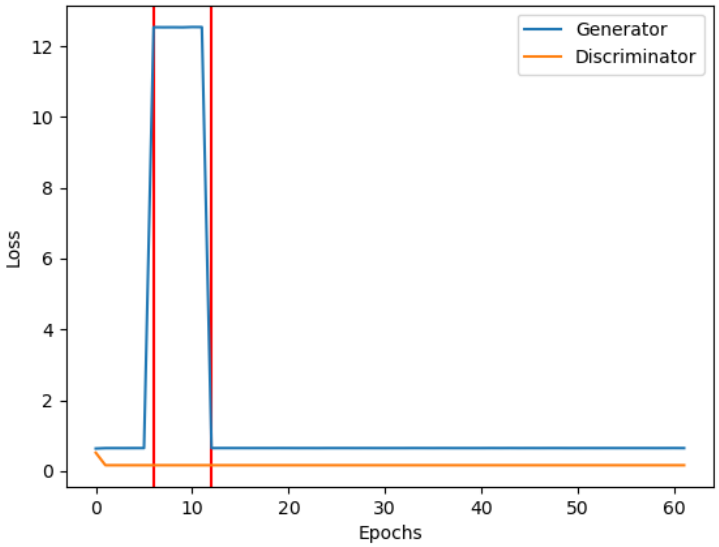


Figure 5.6.: E2 – Generator and discriminator loss.

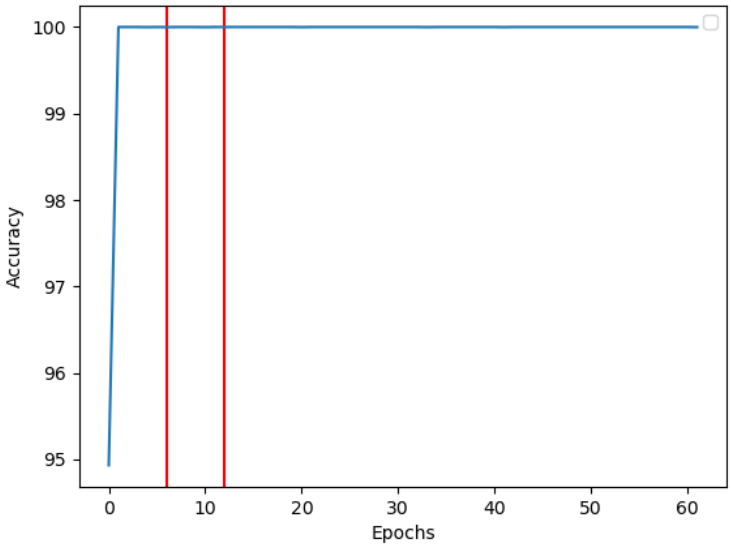


Figure 5.7.: E2 – Discriminator accuracy.

Figure 5.6 shows the loss of G and D when trained for 50 epochs, plus six pretraining epochs for both, on 100 samples with these techniques implemented. The red lines marks the end of the pretraining epochs. For the first six epochs, G is frozen, and D is pretrained on recognizing the generations of G. After six epochs, marked by the first red line, D is frozen and G unfrozen. G is then pretrained for six epochs, until the second red line when both G and D are unfrozen. Figure 5.7 shows the accuracy of D. It still appears that the discriminator becomes too strong compared to the generator, resulting in a vanishing gradient. The loss of G benefits greatly from the implemented techniques, with the loss becoming much smaller than the loss in experiment 1.

5.2.4. Results of experiment 2

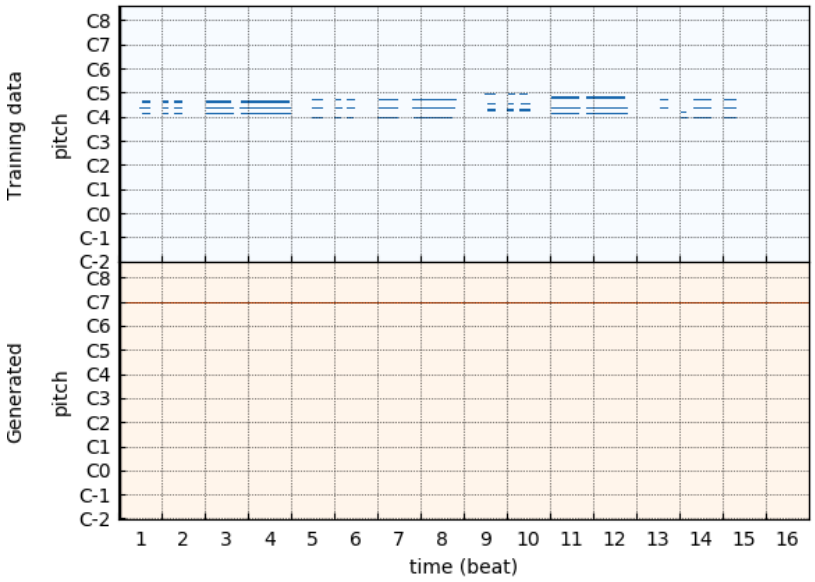


Figure 5.8.: E2 – Generated accompaniment and input melody.

When viewing the generated accompaniments from this experiment, it becomes apparent that although the loss of the generator has decreased substantially, the generated accompaniments all still comprises of a single held note for all inputs. This is also reflected in the accuracy of the discriminator, which becomes 100% after a few epochs. Figure 5.8 presents an example generation along with the leading melody on which it was based on.

5.2.5. Experiment 3: Learning rates

The learning rate of both G and D was set to 0.1 in experiments 1 and 2. This learning rate may be too high for G to learn the data distribution, causing G to output the same accompaniment for all inputs. This could also lead to D learning the difference between real and fake too fast. Therefore, some experiments with decreasing the learning rate to 0.01, 0.005 and 0.001 were conducted. The techniques implemented in experiment 2 are still used. The results of these experiments are presented in the figures below. The losses and accuracy during pretraining for D and G are not included in the figures.

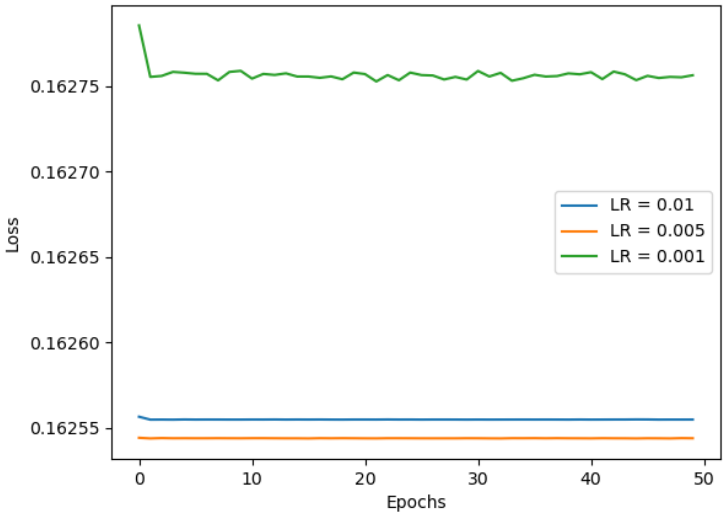


Figure 5.9.: E3 – Discriminator loss with different learning rates.

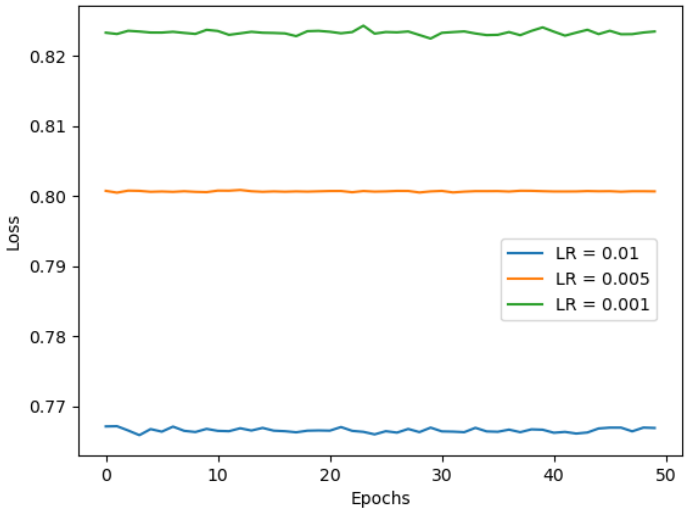


Figure 5.10.: E3 – Generator loss with different learning rates.

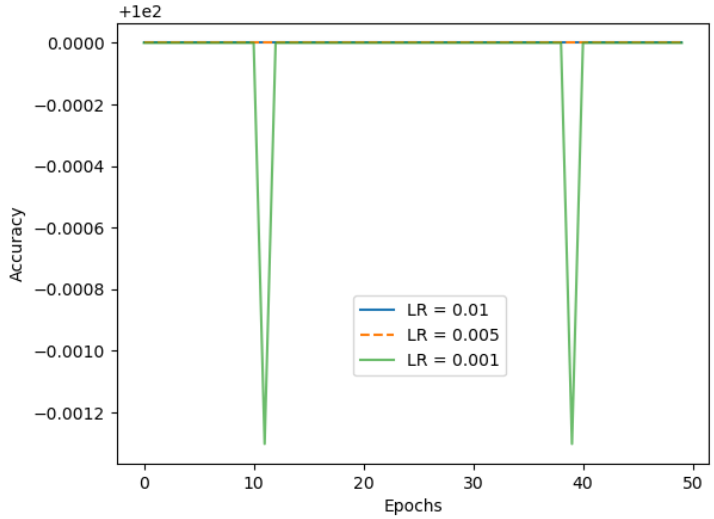


Figure 5.11.: E3 – Discriminator accuracy with different learning rates.

Figure 5.9 shows that the learning rates did not have a major impact on the losses of D and G. For learning rates 0.01 and 0.005, the accuracy of the discriminator is 100%. D started showing a slight decrease in accuracy when the learning rate was set to 0.001. The generated accompaniments for this learning rate also display some degree of note variation. 0.001 was therefore chosen as the learning rate for the experiments to follow.

5.2.6. Results of experiment 3

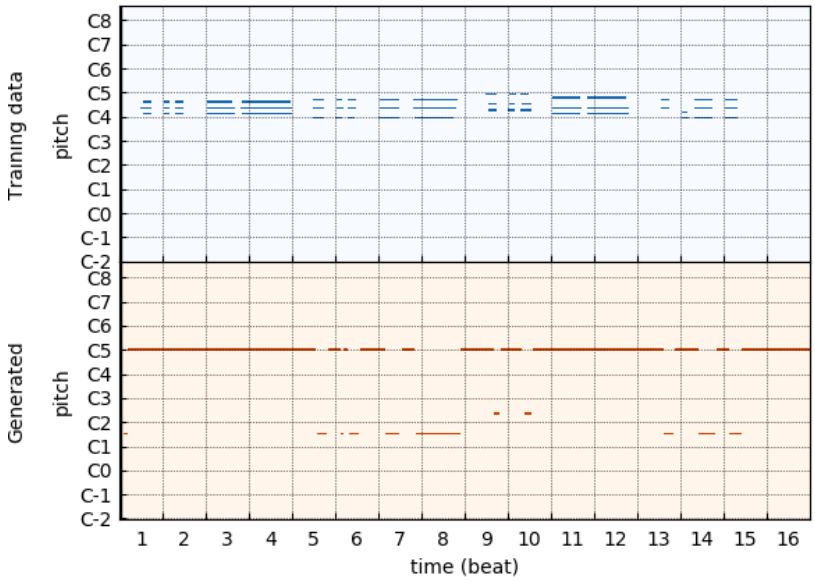


Figure 5.12.: E3 – Generated accompaniment and input melody with learning rate 0.001.

The learning rates 0.01 and 0.005 displayed similar behaviour as in the previous experiments, with the same single note being generated for all inputs. Figure 5.12 presents an example of a melody and the generated accompaniment for the model with learning rate 0.001. The generated accompaniment contains three unique notes of different durations. The generated accompaniment is not in the same key as the leading melody. If the span of the notes is disregarded, as well as the key of the melody, this could in isolation be considered a valid accompaniment. However, by looking at the other generated accompaniments for this model, it becomes clear that the generated accompaniments are very similar, and does not seem to adapt to what is being played by the leading melodies. Figure 5.13 presents another generated accompaniment from this experiment which displays this behaviour.

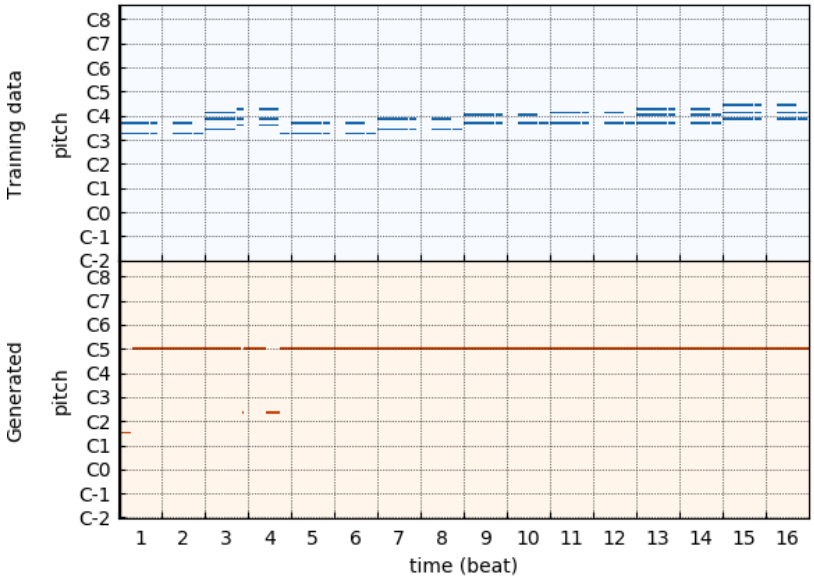


Figure 5.13.: E3 – Another generated accompaniment and input melody example.

The generated accompaniment presented in figure 5.13 is very similar to the one in 5.12. They contain the same three notes, even though the leading melodies for these generations are in different keys. By listening to the combined MIDI files for these samples, it becomes apparent that the generated accompaniment does not fit with the leading melody, and is often unpleasantly dissonant.

5.2.7. Experiment 4: Multiple time steps as input

In experiments 1, 2 and 3, the number of time steps that was used as input for G and D was one. This means that the model only got the previously generated time step, and the current time step in the input phrase, as input. When there are 384 time steps for an 8-bar phrase, each time step is a 48th note. Musical phrases rarely feature 48th notes. More commonly used note durations are for example whole notes, quarter notes, 8th notes etc. An 8th note in the training data would be represented as six identical time steps after one another. Tables 4.2 and 4.3 on page 29 showed that the average durations of the guitar and bass phrases in the training data was 8.8 for the guitar, and 12.2 for the bass. By inputting multiple time steps, it was believed that G would be able to better understand the variations and duration of notes in the input phrases. With six time steps being used as input, the initial time steps g_{-6} to g_{-1} and t_{-5} to t_{-1} were empty arrays, with t_1 being the first time step in the input phrase. These time steps were then concatenated together, so the input into G became $12 \times 84 = 1008$ features. D was also modified in order to account for multiple time steps. D looked at the same

number of time steps as G did during the generation, in order to make its judgment on whether an accompaniment was real or fake. The hidden dimension of both D and G was increased to 512 for this experiment, due to the increase in input features.

In this experiment, the number of time steps that were tested in the sliding window was 6, 12, 24 and 48. These sizes were chosen because they represent commonly used note durations. 6 time steps is an 8th note, 12 is a quarter note and so on. This makes it possible to represent up to whole notes in the sliding window.

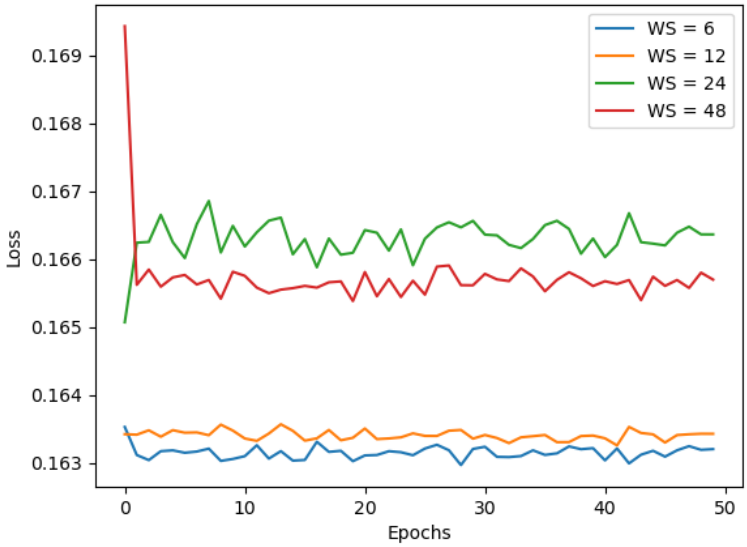


Figure 5.14.: E4 – Discriminator loss with different window sizes.

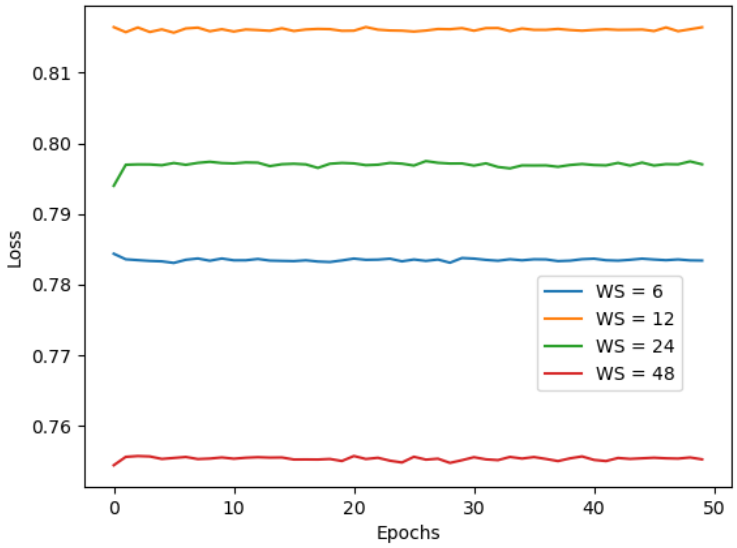


Figure 5.15.: E4 – Generator loss with different window sizes.

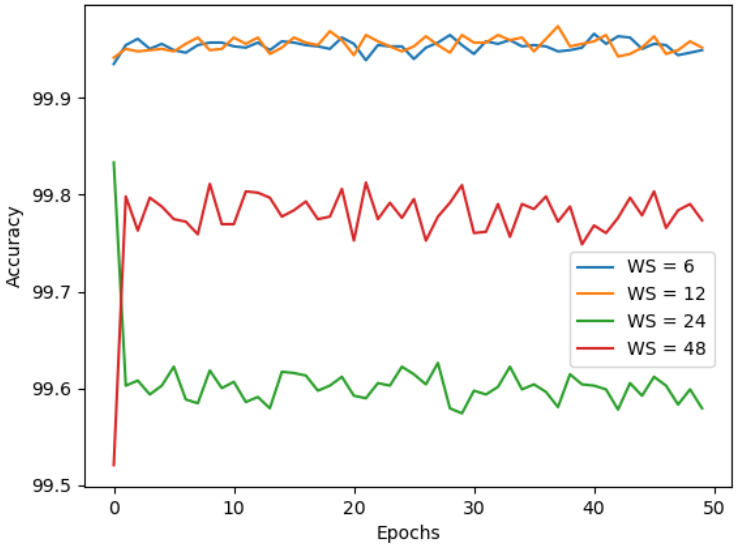


Figure 5.16.: E4 – Discriminator accuracy with different window sizes.

As shown in figure 5.16, the larger window sizes results in a decrease in the accuracy of D. The loss of D also decreases with the size of the window size with window size 24 showing the lowest accuracy. The loss of G is lowest when the window size is 48. The

generated accompaniments from the different window sizes also show that the variety of played notes increases when the window size is expanded. Although window size 24 displayed the lowest discriminator accuracy, viewing the generated accompaniments showed that the accompaniments were two notes that were changed between at seemingly random times. Window size 48 displayed more variation, and more consistent musical ideas and was therefore chosen as the window size for the experiments that followed.

5.2.8. Results of experiment 4

Generated accompaniments from the tests in experiment 4 displayed more variations than the preceding experiments. By inputting multiple time steps to the model, the idea was that the generated accompaniments would generate more varied generations, by allowing the generator to pick up on the different note variations and their duration. There was still a large span between the generated notes in this experiment for all window sizes, except size six, where the output was a single held note for all inputs.

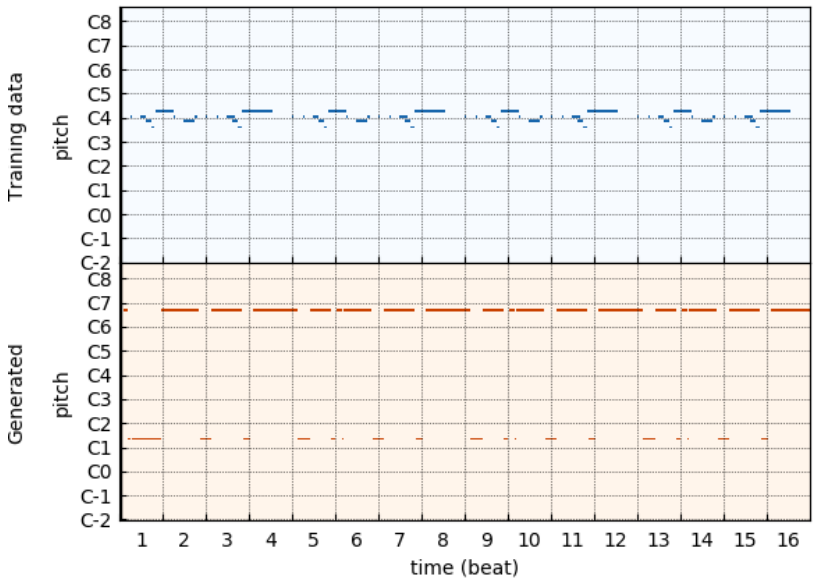


Figure 5.17.: E4 – Generated accompaniment and input melody with window size 24.

Figure 5.17 presents a leading melody and its generated accompaniment for window size 24. There are only two unique notes being played throughout the accompaniment, but there are more frequent shifts between them than in previous experiments. Also, the duration of the notes is shorter, which is closer to the real accompaniments. The generated accompaniments are all very similar, with the same two notes being in the generation regardless of the input.

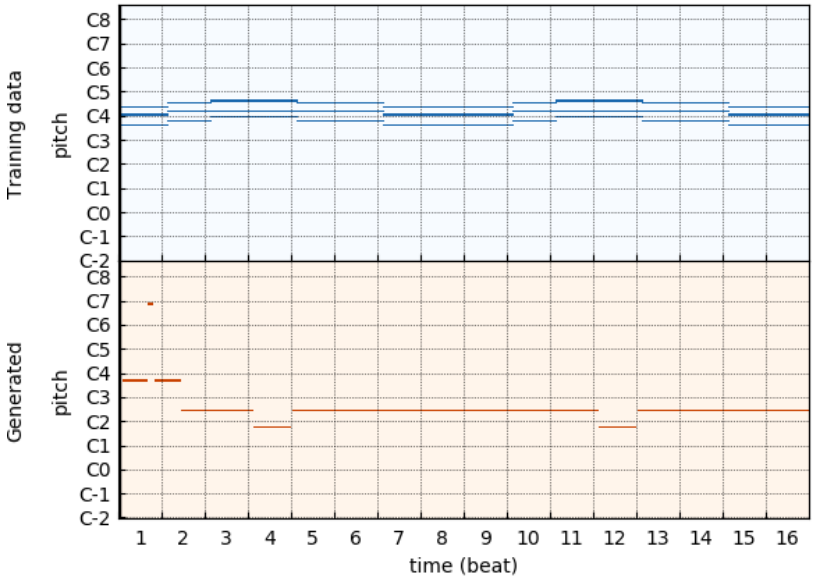


Figure 5.18.: E4 – Generated accompaniment and input melody with window size 48.

When using window size 48 there appears to be more variation in the accompaniments, and a more coherent musical ideas. The main problem is still that the generated accompaniments are all very similar. They all consist of a total of five unique notes. This indicates that the system is not capable of finding which notes that fit with the currently played notes in the leading melody. There are however some interesting features of the generated accompaniment. When viewing figure 5.18, at time beat 4 there is a change in the accompaniment. This change can also be observed at time 12. When looking at what is being played in the melody at that time, it can be seen that it is the chord. This indicates that the system learns to change the note based on the current events in the leading melody. This behaviour can also be observed for other generated accompaniments. So it seems like the model is reacting to what is being played in the melody, but has a limited vocabulary as to which notes it can play.

5.2.9. Experiment 5: Reducing the tonal range of time steps

The samples generated in experiment 4 indicates that the system starts to have more note variety the more time steps that are in the sliding window. One issue with these generated accompaniments is that the span between pitches is very large, with some exceeding the register of a regular bass. Another issue with inputting more time steps is that the number of features becomes large. As seen in experiment 4, using 6 time steps as input makes the inputted features 1008. By reducing the tonal range to 24, instead of 84, the inputted features with a window size of 6 time steps is $12 \times 24 = 288$. The

number 24 was chosen because it is equivalent to two octaves. This was also suggested in the preceding project (Berg, 2019). As shown in the tables 4.2 and 4.3 in section 4.1, the average span for the guitar and bass phrases in the training data is 14.8 for the guitar, and 12.8 for the bass. The reduction in size is done using a modulo operation on indices of the notes being played in the time step. So, for example the C-major chord "24-28-31" would become "0-4-7" after the modulo operation.

One aspect of the training data is that pitches are considered separate entities. For example the pitches C2 and C4 have no explicit connection in the data. By using the modulo operation on the indices of these pitches, the resulting indices are C2: $12 \bmod 24 = 12$, C4: $36 \bmod 24 = 12$. In other words, phrases in different octaves are transposed into the span of two octaves, giving more explicit information about the pitches used. This only works for pitches that are a multiple of two octaves apart however, since C2 and C3 would get the values 12 and 0 after the modulus operation, which has no explicit connection.

A problem which comes with this approach is that phrases can become jumbled. If a phrase consists of the pitches with indices [21, 24, 30, 27], the resulting pitches after the modulo operation would be [21, 0, 6, 3].

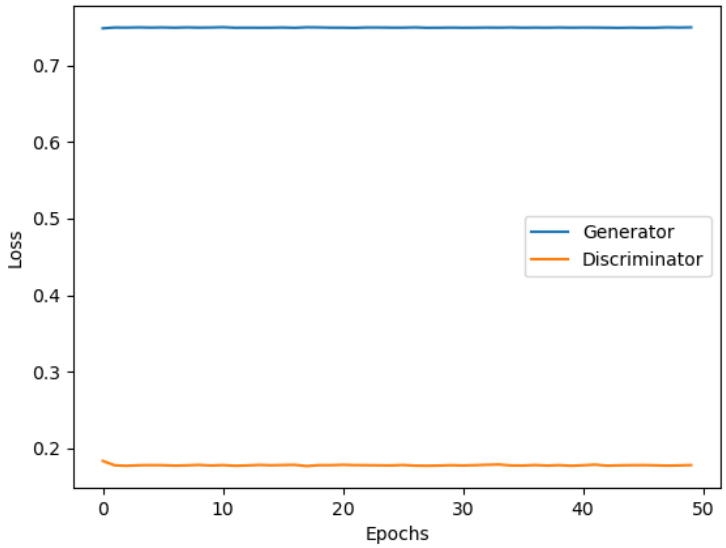


Figure 5.19.: E5 – Generator and discriminator loss with tonal range 24.

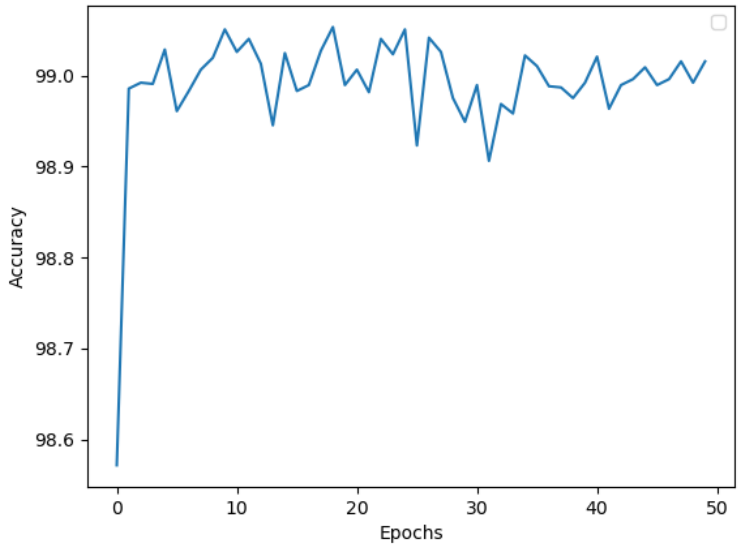


Figure 5.20.: E5 – Discriminator accuracy with tonal range 24.

This approach produced accompaniment with less variations than in experiment 5. Some accompaniments suffered from the same behaviour as described in experiments 1 and 2, where the accompaniments contained the same single note. The other generations displayed some variations of the notes played, but they were very similar to each other regardless of input. As seen in figure 5.20, the accuracy of the discriminator fluctuates around 99 %. This experiment, like the previous ones, indicates that D becomes too strong compared to G.

5.2.10. Results of experiment 5

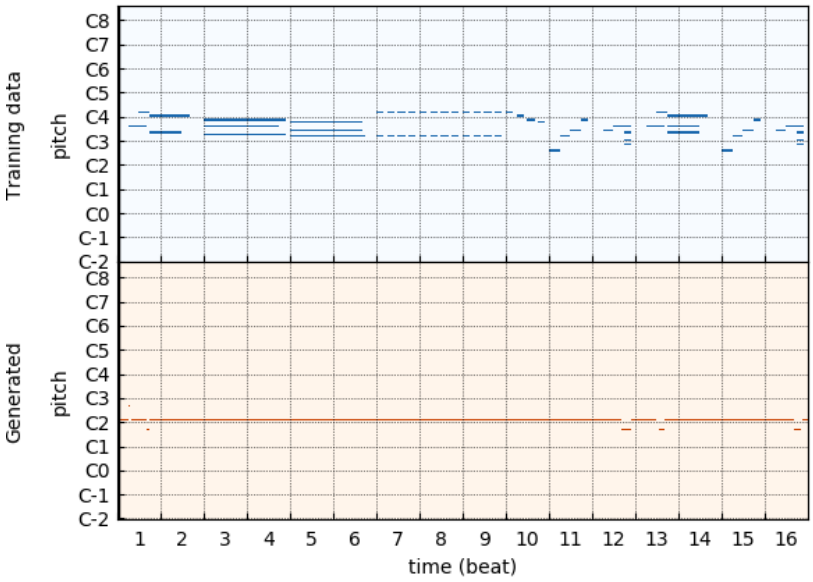


Figure 5.21.: E5 – Generated accompaniment and input melody with reduced tonal range.

In experiment 5, the tonal range was reduced to 24. This was done in order to combat the problem regarding the span in pitches in the preceding experiments. Figure 5.21 presents a sample generation from this experiment. Some of the generations for this experiment were a single note held for all 384 time steps. The sample generation displays very little variation. A total of three notes is played throughout the phrase, with one note having a very long duration whenever it is played. This reduction in tonal range displayed the same behaviour as seen in the previous experiments. The generated accompaniments all consist of the same limited number of notes being played. There are certain variations between the generated accompaniments, but they are very similar. As in the previous experiment, with window size 48, there appears to be some change in the generated accompaniments based on what is being played in the leading melody. At time 12 in figure 5.21, there is a small motif being played by the leading melody. This is repeated at time 16. At both these times, the accompaniments changes the note that is being played. This is however observed infrequently, as two of the accompaniments consist of only a single held note.

5.2.11. Experiment 6: Only melody as input

In this experiment, only the leading melody was used as input. This means that the previous generated time steps by the G is not taken into consideration when G makes its next generation. The motivation behind this experiment is that in the previous experiments, there was a small of degree of variation across the generated accompaniments. Many of the generated accompaniments appeared to not react to the events occurring in the leading melody. When using only the leading melody as input, the system is forced to base its generation on just the leading melody.

For this experiment, a window size of 48 was used. Both the full and reduced tonal range were tested. The models were only run for 20 epochs, due to a limitation in time and computational power. From the previous experiments, a greater number of epochs did not seem to improve the results of the model, so this is believed to not have much of an impact on the results of this experiment.

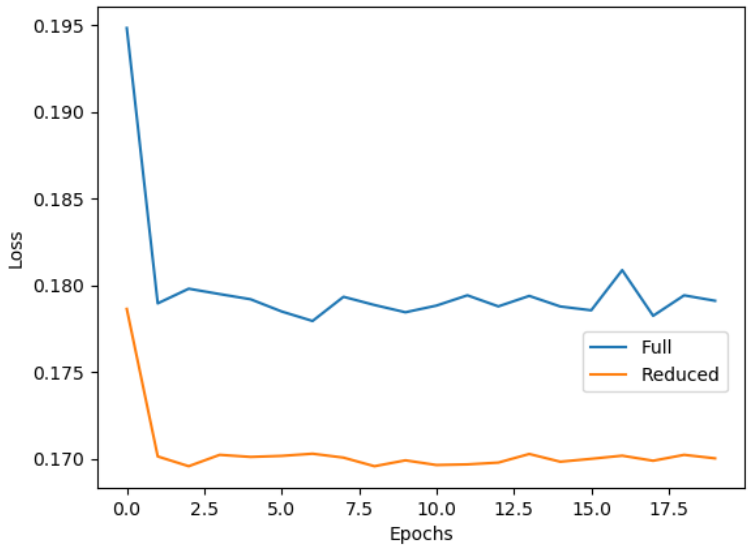


Figure 5.22.: E6 – Discriminator loss with different tonal ranges.

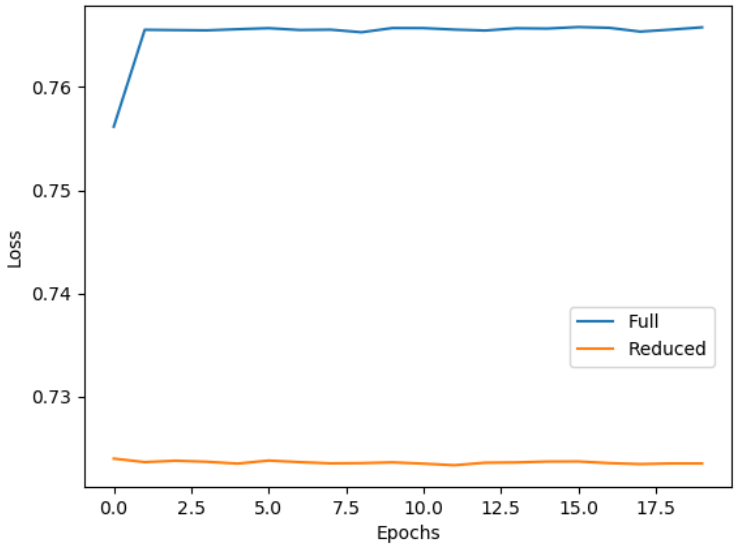


Figure 5.23.: E6 – Generator loss with different tonal ranges.

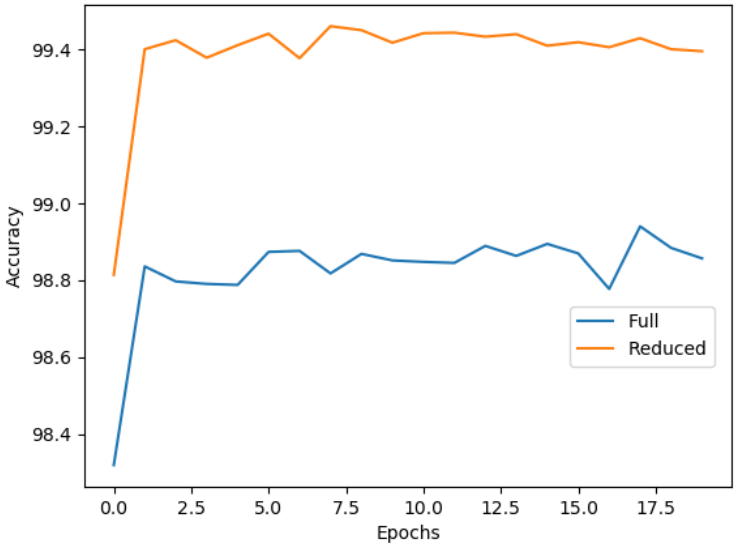


Figure 5.24.: E6 – Discriminator accuracy with different tonal ranges.

Figure 5.24 shows the accuracy of D for both the full and reduced tonal range in the time steps. The full tonal range seems to produce accompaniments that fools D more often than the reduced tonal range. The generated accompaniments from the full

tonal range displayed a greater variation across the generated accompaniments than in previous experiments. By reducing the tonal range of time steps, the variation between the generated accompaniments were also reduced.

5.2.12. Results of experiment 6

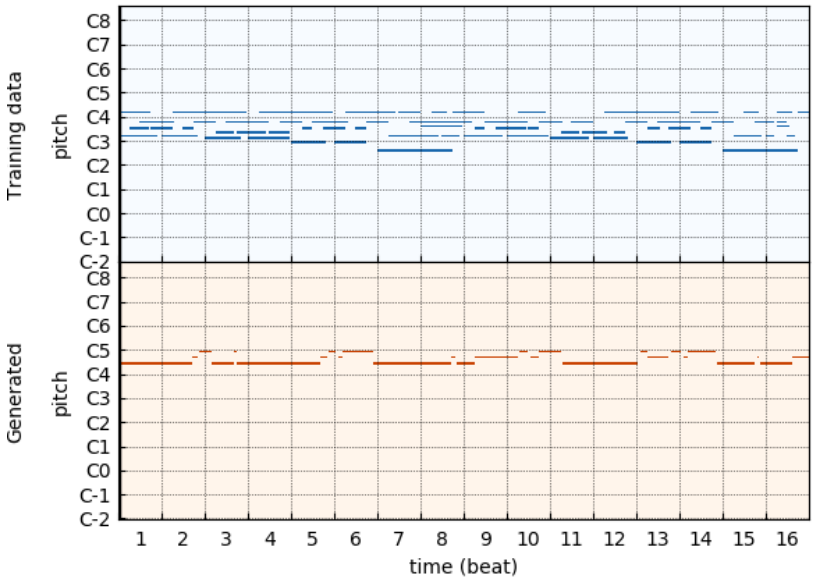


Figure 5.25.: E6 – Generated accompaniment and input melody with full tonal range.

Experiment 6 changed the input data of G, so that the generated accompaniments are based on just the leading melody. This is what is described as C2 in 4.2.1. Figure 5.25 shows that the generated accompaniment displays greater degrees of variation than in the previous experiments. There are more changes in the notes played, and when comparing all the generated accompaniments, it becomes clear that they are more unique and distinct than in previous experiments. There are also repeating motifs in the accompaniments. The total note vocabulary across the generated accompaniments appears to limited, as there are only four unique notes across all generated accompaniments for this configuration. The problem regarding the wide span of notes played is not present in this configuration, so it appears that the system learns to avoid it. When using the full range of notes however, it appears that the generated accompaniments are played in an uncommon tonal range for bass accompaniments. A normal 4-string bass guitar with 22 frets has a tonal range of E1 - F4. The generated accompaniments are played in the range D♯4 - A4, which is a tonal range that is not possible to play in for a normal bass guitar tuned to standard tuning.

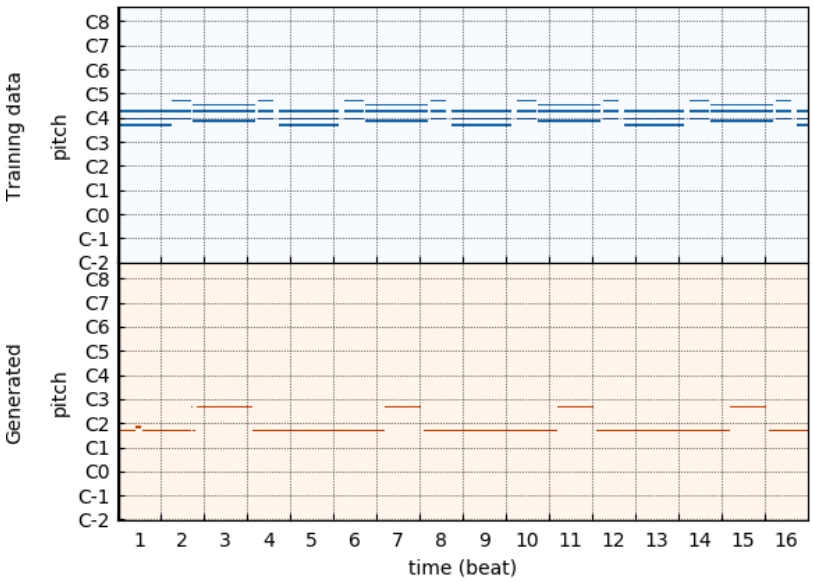


Figure 5.26.: E6 – Generated accompaniment and input melody with reduced tonal range.

The problem regarding the tonal range of the accompaniments is not present with the reduced range of pitches, since it is limited to a range of C1 - B2. Figure 5.26 presents a sample generation which showcases this behaviour. This reduction in tonal range caused there to be less variation between the accompaniments than the full tonal range configuration. The problem regarding the limited note vocabulary was still present for this configuration. The accompaniment in figure 5.26 does however react to the events of the leading melody. There is a clear motif being repeated through the accompaniment, which fits with the repetitions in the leading melody.

5.3. Statistical overview

This section presents the statistics of the generated accompaniments from the experiments in the previous sections, and compares them to the statistics of the accompaniments in the training data. For the experiments 1 and 2, the generated samples were all the same single note held for the duration of the phrase, regardless of input. The accuracy of the discriminator D was also 100%. This is evidence of the model suffering from mode collapse and a vanishing gradient for the generator G. These results will therefore not be discussed further in this section. In experiment 3, the generated samples from the different learning rates also suffered from this, with the exception being when the learning rate was 0.001. With a learning rate of 0.001, some note variations in the generated accompaniments were observed. This was also reflected in the accuracy of D, which was observed to be slightly lower. Therefore, 0.001 was chosen as the learning rate for the experiments that followed. Experiment 4, 5 and 6 showed some note variations across the generated samples, and the results of these experiments are discussed in this section.

One important thing to note here is that GANs do not try to reproduce the training data. Instead, the goal is to generate new data that is supposed to be indistinguishable from the training data. Therefore, comparing the generated accompaniments to the real phrases is not enough to give the final evaluation of the systems ability to generate accompaniments. It does however give a general overview of how the generated accompaniments differ from the real accompaniments, which can be part of the evaluation.

In experiment 4, the number of time steps that was used as input to the model was tested. By implementing a sliding window technique for the input, the model is able to use several time steps in order to make its generation. The tested window sizes were 6, 12, 24 and 48. The statistics for the five generated accompaniments, as well as the statistics of the real accompaniments, are presented in the tables 5.2, 5.3, 5.4 and 5.6.

Window size 6						
Generated accompaniment statistics:						
	Pauses	Chords	Notes	Uniques	Durations	Spans
Max	0	0	384	1	384	0
Min	0	0	384	1	384	0
Average	0	0	384	1	384	0
Real accompaniment statistics:						
	Pauses	Chords	Notes	Uniques	Durations	Spans
Max	192	1	358	11	23	17
Min	25	0	192	5	1	5
Average	83.8	0.4	299.8	8.6	6.6	11

Table 5.2.: E4 – Statistics for window size 6.

With window size six, all the generated accompaniments were a single note held for the whole duration of the phrase. This note was the same regardless of input. This is

reflected in the statistics in table 5.2, where the average number in the Uniques column for the five generated samples is one, and the average duration of phrases is 384 time steps.

Window size 12						
Generated accompaniments statistics:						
	Pauses	Chords	Notes	Uniques	Durations	Spans
Max	4	0	383	4	279	49
Min	1	0	380	3	1	39
Average	2.8	0	381.2	3.4	76.4	41
Real accompaniments statistics:						
	Pauses	Chords	Notes	Uniques	Durations	Spans
Max	135	57	367	21	96	19
Min	11	0	249	6	1	7
Average	59	11.4	313.6	10.6	13.3	13.6

Table 5.3.: E4 – Statistics for window size 12.

When the window size is expanded to 12 time steps, there is an increase in the variation of notes played. The statistics in table 5.3 show that there is on average 3.4 unique events in the generated accompaniments. One thing to note in these statistics is the pauses. These are the results of how pypianoroll inserts pauses between event changes. The model itself is incapable of generating pauses. An interesting statistic here is the span between the highest and lowest notes. The minimum span in the generated accompaniments is 39, which is equivalent to over three octaves. This is much greater than the span in the real accompaniments, where the maximum span is 19. In fact, a span of 39 semitones is a greater span than the span of a normal bass guitar with 22 frets and 4 strings in standard tuning, which is only capable of 37 semitones (E1 to F4). On average, the span of the generated accompaniments is over three times the span of the real accompaniments.

Window size 24						
Generated accompaniments statistics:						
	Pauses	Chords	Notes	Uniques	Durations	Spans
Max	17	0	383	3	381	64
Min	1	0	367	3	1	64
Average	7.4	0.0	376.6	3.0	45.64	64
Real accompaniments statistics:						
	Pauses	Chords	Notes	Uniques	Durations	Spans
Max	58	3	377	12	48	14
Min	7	0	326	6	2	10
Average	28.2	355.2	381.2	8.4	13.4	11.8

Table 5.4.: E4 – Statistics for window size 24.

Increasing the window size to 24 time steps leads to even more variations in the generated accompaniments. The average number of pauses increases, as well as a decrease in the average duration, indicating that there are more note changes in the accompaniment. The span of the generated accompaniments increases when using this window size, meaning that the accompaniments contain notes from different octaves. This is a much greater span than what is possible on a normal bass guitar. The maximum span of bass parts in the training data, as presented in table 4.3, is 43, which means that the span in the generated accompaniment is nearly two octaves greater than the maximum span of bass parts in the training data. When comparing the generated accompaniments to the real accompaniments for the melodies used in the generative process, the span is over five times greater.

Window size 48						
Generated accompaniments statistics:						
	Pauses	Chords	Notes	Uniques	Durations	Spans
Max	11	0	381	5	339	61
Min	3	0	373	5	1	61
Average	6.6	0	377.4	5.0	34.2	61
Real accompaniments statistics:						
	Pauses	Chords	Notes	Uniques	Durations	Spans
Max	99	4	380	8	48	12
Min	0	0	285	5	1	7
Average	55.6	1.0	327.4	6.2	13.7	8.4

Table 5.5.: E4 – Statistics for window size 48.

Window size 48 produces samples with more unique events, as can be seen in the Uniques column. This is however only slightly more than 10% of the unique events in the real data. The span with window size 48 is 61 for all generated samples, which seems to indicate that increasing the window size does not help in reducing the span between the different notes. This is the motivation behind experiment 5, where the range of 84 possible notes for each time step is reduced down to 24.

Window size 48, tonal range 24						
Generated accompaniments statistics:						
	Pauses	Chords	Notes	Uniques	Durations	Spans
Max	6	0	384	4	384	12
Min	0	0	378	1	1	0
Average	2.2	0	381	2.6	173.3	7.6
Real accompaniments statistics:						
	Pauses	Chords	Notes	Uniques	Durations	Spans
Max	141	2	362	10	96	15
Min	22	0	243	7	1	5
Average	81	0.4	302.6	8.4	8.5	12.8

Table 5.6.: E5 – Statistics for window size 48 and reduced tonal range.

By reducing the tonal range to 24, the span of the generated accompaniments were reduced down to a maximum value of 12. The minimum value of the span is 0, which means that some of the generated accompaniment consist of only a single held note. The main discrepancies between the generated and real accompaniments is the durations and pauses. The real data having more pauses is to be expected, due to the system's inability to generate pauses. This could also explain why the durations are so different. The system is unable to generate phrases where for example, two half notes of the same pitch are played in succession. The system would generate this as one whole note. This could be resolved by implementing some kind of post-processing of the generated accompaniment, by inserting pauses if a note is held for longer than for example 24 time steps, but this has not been implemented in the system in this thesis.

Window size 48, full tonal range						
Generated accompaniment statistics:						
	Pauses	Chords	Notes	Uniques	Durations	Spans
Max	15	0	383	5	339	19
Min	1	0	369	4	1	6
Average	10	0	374	4.2	37	8.6
Real accompaniment statistics:						
	Pauses	Chords	Notes	Uniques	Durations	Spans
Max	192	1	358	11	23	17
Min	25	0	192	5	1	5
Average	49.2	1	333.8	8.6	20.3	10.2

Table 5.7.: E6 – Statistics for window size 48 and full tonal range.

Window size 48, reduced tonal range						
Generated accompaniment statistics:						
	Pauses	Chords	Notes	Uniques	Durations	Spans
Max	5	0	384	4	314	12
Min	0	0	379	2	1	12
Average	2.6	0	381.4	3.2	44.8	12
Real accompaniment statistics:						
	Pauses	Chords	Notes	Uniques	Durations	Spans
Max	170	4	381	16	48	22
Min	0	0	210	4	1	4
Average	90.2	1.4	292.4	10	9.6	12.8

Table 5.8.: E6 – Statistics for window size 48 and reduced tonal range.

The statistics for experiment 6 are presented in table 5.7 and 5.8. With the full tonal range, the average number of pauses increases, indicating more note changes throughout the generated accompaniments. Also, the span of the generated accompaniments have been significantly reduced from the previous experiments. In general, all of the statistics in table 5.7 display more variations than in the previous experiments, indicating that there are more distinct generated accompaniments with this configuration. The number of uniques is however still fairly low compared to the uniques of the real accompaniments. The reduced tonal range displayed less variation than the full, as displayed in table 5.8. The different statistics of the generated accompaniments are less varied, with all accompaniments having a span of 12.

6. Evaluation and Discussion

This chapter contains the evaluation of the system, as well as the discussion of the results of this thesis with regards to the research goals and research questions.

6.1. Evaluation

This chapter evaluates the results from the experiments conducted in chapter 5. The first section is an objective evaluation of the system's ability to generate accompaniments that are within the statistical boundaries of the training data. The second section presents a subjective evaluation of the generated accompaniments.

6.1.1. Objective evaluation

The statistics presented in section 5.3 reveal that there are many differences between the real and generated accompaniments. Some of these differences are due to the system's inability to generate pauses, which would decrease the duration of notes and increase the amount of pauses in the accompaniments. When comparing the statistics of the generated accompaniments to the statistics of the training data in tables 4.2 and 4.3 on page 29, a case could be made that some of generated accompaniments could be considered valid. The statistics shows that the different features of the training data are very diverse. For example, there are phrases where a single note is held for the duration of the phrase for both the bass and guitar parts. There are also phrases that are predominantly pauses, which the system is not able to generate.

The tonal range in phrases ranges from 0 to 56 for guitar parts, and 0 to 43 for the bass parts. Comparing all the statistics from the tables to the statistics of the generated accompaniments, it appears that some of the generated accompaniments from the experiments fall within the minimum and maximum bounds of the different statistics. Therefore, the system can be said to generate some accompaniments that are within the statistical bounds of the training data. However, this conclusion ignores some important limitations of the generated accompaniments.

Experiment	Learning rate	Window Size	Reduced	Similarity
1	0.1	1	False	1.0
2	0.1	1	False	1.0
3	0.01	1	False	1.0
3	0.005	1	False	1.0
3	0.001	1	False	0.85
4	0.001	6	False	1.0
4	0.001	12	False	0.75
4	0.001	24	False	0.45
4	0.001	48	False	0.84
5	0.001	48	True	0.93
6	0.001	48	False	0.33
6	0.001	48	True	0.82

Table 6.1.: Similarity across the five generated accompaniments for every experiment.

Table 6.1 displays the overall similarity across the generated accompaniments for all experiments. The similarity is calculated by comparing the time steps of each generated accompaniment to all the others, and taking the average similarity between all of them. If all five generated accompaniments for an experiment were the same, the similarity score would be 1.0. By viewing the table, it becomes evident that the majority of the experiments generates accompaniments that all are very similar. The full tone range configuration from experiment 6 has the lowest similarity score of 0.33. This means that on average, 33% of all the time steps in the five generated accompaniments from this experiment were identical. As described in 5.3, this configuration displayed more variations across all generated accompaniments. It also displayed behaviour of playing repeating motifs based on the leading melody, which can be seen in figure 5.25. There are however two main issues with this configuration. It has only a limited set of notes that are played across all generated accompaniments, and the register the accompaniments are played in is outside the range of a normal bass guitar. The reduced tonal range configuration handled the issue regarding the register, but this caused the variation in generated accompaniments to decrease, as seen by its similarity score in table 6.1. This configuration also had a limited note vocabulary. One thing to note in the table is window size 24 having a similarity of 0.45. This is deceptively low, as a visual inspection of the generated accompaniments reveals that the accompaniments are all the same two notes being changed between at seemingly random times.

By using just the leading melody as input, variation of notes in the generated accompaniment increases leading to more distinct accompaniments. This however causes the accompaniments to be played in an odd register for bass accompaniments. By reducing the tonal range to a more common bass register, the variation of notes across the accompaniments decreases leading to more similarity in the generated accompaniments. This is seen in table 6.1, where the similarity across all generations of the reduced time step span is 0.82.

6.1.2. Subjective evaluation

In the project preceding this thesis, it was proposed that a subjective survey should be conducted in order to evaluate the generated accompaniments. This was also the plan at the start of writing this thesis. However, due to the aforementioned limitations of the generated accompaniments, this was deemed unnecessary. The system generates accompaniments that display some degrees of variations, but tend to not fit with the notes in the leading melody. This becomes very apparent when listening to the generated samples. Almost all generated accompaniments sound dissonant against the leading melody. The majority of those that do sound somewhat fitting are fairly uninteresting, and their fitness is most likely due to coincidence, because of the limited note vocabulary discussed earlier. Another aspect is that the system is incapable of generating accompaniments that contain pauses. A subjective study could be conducted where the generated accompaniments are transposed to the correct key. Also, the pauses of the training data could be removed, so that the presence of pauses would not reveal that the accompaniment was real. Another approach is to add pauses to the generated accompaniment, for example if the duration of notes is longer than a whole note.

This approach however was deemed to involve too much human interference to be a genuine evaluation of the produced accompaniments. Adding or removing pauses from the accompaniments is a valid technique, since the generator is not capable of generating pauses. But the transposition of accompaniments so that they fit with the leading melody seems to contradict the overall purpose of the system. A system which generates accompaniments should be able to generate accompaniments that contain notes that fit with leading melody. This requires the system to be able to recognize which notes do and do not fit with what is currently being played. By transposing the accompaniments to the right key, the act of making the accompaniment fit with the leading melody is done by the human agent and not the system.

There are however some generated accompaniments that fit with the leading melody, and are played in the correct key. That the accompaniment is in the correct key is most likely a coincidence due to the aforementioned limited vocabulary present in all generated accompaniments. Figure 5.26 on page 59 displays an example of a generated accompaniment that fits with the leading melody, and sounds quite musical. This is likely due to the key of the leading melody containing the limited number of notes in the system's vocabulary. Comparing this accompaniment to the other generated accompaniments from the same configuration shows that they share the same notes, but that this one is somewhat distinct and fits with the leading melody.

Across all the generated accompaniments from all experiments, there appears to be an issue regarding the limited note vocabulary the system obtains. This means that the system does not appear to understand the concept of keys and the relationship between notes. In some of the experiments, the system displayed behaviour of reacting to events in the input data, but it did not appear to understand which notes fit with the notes in the leading melody. For the vast majority of the generated accompaniments, the subjective evaluation is that they do not sound like they fit with the generated accompaniments.

6.2. Discussion

This section presents a discussion of the work and results of the experiments conducted, and how these relate to the overall goal and research questions of this thesis.

6.2.1. Goal: Explore the capabilities of Generative Adversarial Network consisting of Recurrent Neural Networks in the field of automatic accompaniment generation

The architecture that was implemented in this thesis does generate accompaniments, and the generated accompaniments do exhibit some variations between them. However, the results presented in section 5.3 show that the system tends to generate accompaniments that do not fit with the leading melody. The first five experiments resulted in accompaniments with little variation across all generations. The sixth experiment displayed more distinct accompaniments for the leading melody, but they all shared the same limited vocabulary of notes. The sixth experiment also displayed the greatest capability of reacting to the changes in the leading melody. The limited vocabulary of notes is a problem which results in many of the generated accompaniments sounding dissonant, and not fitting with the leading melody. The system seems to not be able to learn which notes fit together, and does not recognize the key of the leading melody. It does however seem to react to changes in the leading melody, and creating corresponding motifs for repeating motifs in the melody. This is an interesting and encouraging aspect of the system, which should be researched further.

The system focuses on generating bass accompaniments for the leading melody, as this was deemed to be an easier task than for example piano accompaniment. Although the system allows for chords to be generated, this was not displayed in any of the generated accompaniments. This is likely due to chords being uncommon in bass parts.

For all experiments, the batch size was set to one. This means that the weights of both the discriminator and generator are updated after each sample in the training data. Increasing the batch size would mean that the number of updates decreases per epoch. This could lead to a more varied vocabulary of notes, because updates would happen less frequently allowing more variations to be generated between updates. A technique called *minibatch discrimination* could also help further increase the variation of the generated accompaniments. The way this technique works is that the discriminator is allowed to view multiple samples in order to make its judgment on whether the samples in the batch are real or fake Salimans et al. (2016). The discriminator has a layer which measures the similarity between the samples in order to aid in the evaluation. This is a technique that is suggested to help with mode collapse. Future work should be done that implements and tests this technique.

The system's inability to generate pauses is also an issue that should be further explored in future work. It was previously suggested that a confidence threshold could be implemented which would allow for pauses to be generated if the most probable note is below the confidence threshold. Another approach is to simply add a pause after a certain duration, for example after whole notes.

Another way of allowing the system to understand how notes fit together is to add some additional information to the system that informs the system of what the nature of each note played is. As the system is now, there is no apparent connection between the same notes in different octaves. By providing some additional information as to the nature of the notes played, the system could better understand how keys and notes relate.

The architecture of the system displays a limited capability of generating accompaniments for melodies in its current state. There is some desirable behaviour exhibited for some of the generated accompaniments, but the limited note vocabulary in all the experiments meant that the generated accompaniments rarely fit the melodies.

6.2.2. Research question 1: How do the generated artefacts compare to the training data?

As described in the section 5.3, there is a distinct difference between the training data and the generated accompaniments. The different experiments produced results that varied greatly, with some producing accompaniments that were more similar to the training data. For example, the generated accompaniments from experiment 5 displayed more variations in the accompaniments compared to the previous experiments. The main differences between these generated accompaniments and the real accompaniments were in the span and variation of notes. With a tonal range of 84, the span of the generated accompaniments far exceed the pitch span of the accompaniments in the training data. Most generated accompaniments from the different experiments exceeds the range of a normal bass guitar, with the exception being when the system produces accompaniments consisting of single notes held for the duration of the phrase, and when the range of notes is reduced to 24. One important thing to note in regards of the models ability to generate fitting accompaniments for a melody is that the system is not capable of generating pauses. This is due to the inherent nature of the system, which wants to give a label each time step. In the training data, a pause is represented as an absence of True values. As suggested earlier, this could be possible to generate by the system if a confidence threshold is implemented. This means that if the probability of the most probable note is beneath a certain threshold, the generated time step is a pause. This would require further experiments to produce the optimal threshold.

The generated accompaniments from experiment six were the ones that were the most similar to the training data. The full tonal range configuration displayed the most variations of notes played of all the experiments. The issue regarding the span from the previous experiments was not present in this experiment. The main difference between the generated accompaniments and real accompaniments were in the tonal range that the generated accompaniments played in. The generated accompaniments were outside the normal register of a bass accompaniment. The reduced tonal range did not have this problem, but displayed less variation of notes played.

In summary, the generated accompaniments is distinguishable from the training data because of a number of specific features. For the majority of the generated accompaniments from the experiments, the span of the generated accompaniments is too great and there is little variety in the notes played. The full tonal range configuration from experiment

6 produced accompaniments that is more varied, but they are all outside the common tonal register of bass accompaniments, making these easily distinguishable from the real accompaniments. The reduced tonal range from experiment 6 displayed less variation, but a tonal range that is more in line with the bass accompaniments in the training data.

6.2.3. Research question 2: Can the generated artefacts be considered valid accompaniments for the leading melody?

Comparing the generated accompaniments to the real accompaniments is not enough to give an accurate evaluation of whether accompaniments can be considered valid accompaniments for a leading melody or not. There can be several fitting accompaniments for a leading melody, with varying statistical features. Music is an audio phenomenon, and the final evaluation of whether an accompaniment can be considered musical is in how it sounds when played alongside the leading melody.

In the introduction to this thesis, it was stated that music is subjective. For some, the generated accompaniments may be considered music. A subjective study was suggested in order to answer this question. This was not completed due the reasons presented in section 6.1.2. The system only obtains a limited note vocabulary, and is not capable of understanding keys and how notes relate to each other. There were also a lot of similarity in the generated accompaniments for some of the experiments. A study could be performed where the generated accompaniments are modified in order to look more like the real training data. This however would require too much human interference in order to be an effective evaluation of the generated accompaniments and the system as a whole. The generated accompaniments do for the most part not fit the leading melody. The ones that do are likely a coincidence of the leading melody being in a fitting key of the notes in the limited note vocabulary that the system obtains.

6.2.4. Research question 3: How do the different system configurations affect the generated accompaniments?

A series of experiments using different configurations for the system were conducted. These experiments involved altering a number of aspects of the system, in order to see the effect it had the generated accompaniments.

The experiments conducted indicates that when lowering the learning rates of the discriminator and generator, the generated accompaniments started to show some note variations. This was further explored by testing the window size of how many time steps that was used as input to the number. This also had an effect on the variations in the generated accompaniments.

In the experiments, it seems that the lower learning rate and higher window size produced more varied results. Though they did not produce accompaniments that could be considered to be very musical, they displayed the most variety of notes as opposed to the other configuration. Reducing the tonal range to 24 resulted in less variety of notes but a more realistic span of notes played.

The final experiment used just the leading melody as input. This resulted in more unique accompaniments that displayed more variations than in previous experiments. The full tonal range configuration in this experiment showcased the most variation, but the resulting generated accompaniments were all played in an odd register for bass. The reduced tonal range generated accompaniments that showed less variations, but they were within a normal bass register. For all configurations, there was an issue regarding the total number of notes that were played for all generated accompaniments. Throughout all the experiments, the system obtained just a limited number of notes in its vocabulary.

6.3. Limitations

Some of the limitations of the system have previously been presented in this thesis. These limitations will be discussed further in this section.

6.3.1. Training data

The training data used for training the system was chosen in order to overcome the problem of time discrepancy between the leading melody and accompaniment in MIDI files. The training data however has some limitations of its own which have been uncovered through the experiments. The main limitation is that there is no inherent connection between the same notes in different octaves, which could be part of the reason for the limited number of notes that the generated accompaniments contain. If some additional information as to which of the twelve notes is in each time step were to be included, it could be easier for the system to understand how notes relate to each other, and generate better fitting accompaniments for the leading melody.

The training data does not have a uniform distribution of phrases in different keys. By transposing all phrases to the eleven other possible keys, it would be possible to obtain a dataset that has no bias towards a certain key. This would however cause the size of the dataset to become very large. Another solution is to choose a key to transpose all the phrases into. For example transposing all phrases in a major key to C and all pieces in a minor key to A. This would ensure that all phrases share the same notes, which could help with the problem of the limited note vocabulary by the system. This would however require there to be some way of calculating the key of each sample in the training data, and transposing it a certain number of steps in order for it to be in the correct key.

6.3.2. Architecture

The architecture has several limitations that have previously been discussed in this thesis. The main limitation is the limited number of notes in the generated accompaniments. At most, the generated accompaniments from the experiments contained four unique notes. These notes were the only ones that were used in the generated accompaniments, regardless of the notes in the leading melody. Some generated accompaniments were only single notes that were held for the whole duration. This is believed to be caused by mode collapse and vanishing gradient, two common problems when training GANs.

Another limitation of the architecture is its inability to generate pauses. As suggested previously in this thesis, this could be remedied by adding a confidence threshold where the most probable note needs to be higher than the threshold in order for the note to be played. If the most probable note is below the confidence threshold, the generated time step would be a pause. Another approach is post-processing, where if a note has a duration longer than a given maximum length, a pause would be inserted.

6.3.3. Experimental process

The experiments conducted also have some limitations that should be highlighted. First of all, the experiments were conducted on 100 randomly selected samples from the training data for each experiment. As seen in tables 4.2 and 4.3, the training data is very varied, and the random selection from this training data may impact the results of the experiments. The batch size for all experiments were set to one, so that the discriminator and generator was updated after each of the 100 samples. With a larger batch size, the updates would be more infrequent, which could lead to different results. Due to limited time and computational power, experiments were not conducted testing these parameters. This was also the cause of the reduction in training epochs for experiment 6. Though the system did not show signs of further improvements with a greater number of epochs, the number of epochs is another parameter that could be tested. Another parameter that was not tested is the number of pretraining epochs for both the generator and discriminator. There are also other parameters such as the number of pretraining epochs and maximum gradient that all could have an impact on the results that were not tested.

7. Conclusion and Future Work

This chapter contains the conclusion of the thesis, as well as the future work, which addresses the limitations of the system and improvements that could be done.

7.1. Conclusion

A generative adversarial network consisting of recurrent neural network was implemented, and its ability to generate accompaniments for melodies was tested. The ability of the system to generate fitting accompaniments for melodies is limited. The generated accompaniments from the first five experiments display little variation in the notes played, and they are all similar. Some desirable behaviour of the system was displayed in some of these accompaniments. The generated accompaniments seem to react to the changes in the leading melody in some of the experiments, which is an important part of a fitting accompaniment. Experiment 6 also displayed a great deal of variation across the generated accompaniments. However, there are some limitations when it comes to the obtained note vocabulary of the system. The system seems to fail to obtain an understanding of the concept of keys, and how notes relate to each other. The generated accompaniments contain the same limited number of notes, regardless of the notes in the input melody. Further research should be done to try and resolve this problem.

In all the experiments, the discriminator shows signs of becoming too strong compared to the generator. This means that the discriminator is able to accurately label accompaniments as real or fake. This causes the gradient used to update the generator to vanish.

Overall, the system displays a limited capability of generating accompaniments for melodies. There are some positive behaviour exhibited, which could serve as a basis on which future research could be built.

7.2. Future Work

Among the future work for this research, the limitations presented in section 6.3 should be addressed. Further improvements should be made in order to combat the problem of mode collapse. Arjovsky et al. (2017) presented an algorithm called Wasserstein GAN, which showed greater stability compared to the regular GAN training. It also showed an ability to combat mode collapse. By implementing this algorithm, some of the limitations of the system could be mitigated.

Minibatch discrimination should also be tested, as it is shown to have a positive effect on mode collapse (Salimans et al., 2016). This technique could also help to increase the note vocabulary of the system.

Adding a confidence threshold for whether notes are played or not should be implemented so that the system is capable of generating pauses. This could help with making the generated accompaniments appear more realistic. Another solution is post-processing, where pauses are added if the duration of notes exceed a given value. Both of these techniques could help make the generated accompaniments appear more realistic.

Another confidence threshold could be implemented in order to generate chords. If for example, the second most probable note is within a given threshold of the most probable note, the generated time step would contain both notes instead of just the most probable. This would be better suited for generating accompaniments for other instruments than bass however, since bass accompaniments rarely contain chords.

The training data does not have an even distribution of phrases in different keys. By transposing each phrase into all different keys, even distribution of keys could be obtained. This would increase the size of the training data substantially. Another alternative is to transpose all phrases into the same key, which could help solve the problem regarding the limited note vocabulary.

The batch size used in all experiments was one. Different sized batches should be tested in the future, as it could also help in increasing the number of notes in the vocabulary. Another potential solution to this problem is to add some additional information to the input into the system, which reveals the nature of the note, helping the system to see the connection between notes in different octaves.

Bibliography

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein Generative Adversarial Networks. 70:214–223, 2017. URL `http://proceedings.mlr.press/v70/arjovsky17a.html`.

Endre Valstad Berg. Automatic Accompaniment Generation. Technical report, Norwegian University of Science and Technology (NTNU), 2019. (Unpublished).

John Biles. GenJam: A Genetic Algorithm for Generating Jazz Solos. *Proceedings of the International Computer Music Conference*, 1994.

Thomas Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, 1996. ISBN 9780195356700.

Simon Colton and Geraint Wiggins. Computational Creativity: The Final Frontier? *Frontiers in Artificial Intelligence and Applications*, 242:21–26, 2012. doi: 10.3233/978-1-61499-098-7-21.

Roger B. Dannenberg. An On-Line Algorithm for Real-Time Accompaniment. *Proceedings of the 1984 International Computer Music Conference*, pages 193–198, 1985.

Roger B. Dannenberg and Bernard Mont-Reynaud. Following an Improvisation in Real Time. *Proceedings of the 1987 International Computer Music Conference*, pages 241–248, 1987.

Roger B. Dannenberg and Gus G. Xia. Improvised Duet Interaction: Learning Improvisation Techniques for Automatic Accompaniment. *Proceedings of the 1987 International Computer Music Conference*, pages 110–114, 2017.

Hélio Magalhães de Oliveira and R. C. de Oliveira. Understanding MIDI: A Painless Tutorial on Midi Format. `https://arxiv.org/abs/1705.05322`, 2017.

Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang. MuseGAN: Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment. *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI)*, 2018a. URL `https://arxiv.org/abs/1709.06298`.

Hao-Wen Dong, Wen-Yi Hsiao, and Yi-Hsuan Yang. Pypianoroll: Open Source Python Package for Handling Multitrack Pianorolls. *Late-Breaking Demos of the 18th International Society for Music Information Retrieval Conference (ISMIR)*, 2018b.

Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks. 12 2016.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Networks. 2014.

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

Alex Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*. Springer-Verlag Berlin Heidelberg, 2012. ISBN 9783642247972.

Lejaren Hiller and Leonard M Isaacson. *Experimental Music: Composition with an Electronic Computer*. McGraw-Hill Book Company, Inc., 1959. ISBN 9789333639659.

Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6:107–116, 04 1998. doi: 10.1142/S0218488598000094.

Andrej Karpathy. The Unreasonable Effectiveness of Recurrent Neural Networks. `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`, 2015. Accessed: 2019-11-11.

Mark Levine. *The Jazz Theory Book*. Sher Music Co., 1995. ISBN 9781883217044.

MIDI Association. MIDI History: Chapter 6 - MIDI Is Born 1980-1983. `https://www.midi.org/articles/midi-history-chapter-6-midi-is-born-1980-1983`, 2015.

Olof Mogren. C-RNN-GAN: Continuous recurrent neural networks with adversarial training. `http://arxiv.org/abs/1611.09904`, 2016.

James Anderson Moorer. Music and Computer Composition. *Communications of the ACM*, 15(2):104–113, February 1972. ISSN 0001-0782. doi: 10.1145/361254.361265.

Michael Mozer. Neural Network Music Composition by Prediction: Exploring the Benefits of Psychoacoustic Constraints and Multi-scale Processing. *Connection Science - CONNECTION*, 6:247–280, 01 1994. doi: 10.1080/09540099408915726.

Barrie Nettles. *Harmony*, volume 1. Berklee College of Music, 1987. URL `https://books.google.no/books?id=pP5eswEACAAJ`.

James R. Norris. *Markov Chains*. University of Cambridge, 06 2012. ISBN 9780511810633.

Chigozie Enyinna Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation Functions: Comparison of Trends in Practice and Research for Deep Learning. `https://arxiv.org/pdf/1811.03378.pdf`, 11 2018.

Chistoper Olah. Understanding LSTM Networks. `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`, 2015. Accessed: 2020-06-19.

Han Qi and Xuan Zou. Automatic Accompaniment Generation with Seq2Seq. `http://qihqi.github.io/machine/learning/music-generation-using-rnn/`, 12 2016. Accessed: 2019-11-08.

Colin Raffel. Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching. `https://colinraffel.com/publications/thesis.pdf`, 2016.

Alan Rich. *Harmony*. Encyclopædia Britannica, inc., 05 2019. `https://www.britannica.com/art/harmony-music/Harmony-and-melody` Accessed: 2019-10-27.

Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Third edition, 2009. ISBN 9780136042594.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *Advances in Neural Information Processing Systems*, pages 2226–2234, 2016.

Marc Schonbrun. *The Everything Music Theory Book: A Complete Guide to Taking Your Understanding of Music to the Next Level*. Adams Media, 2012. ISBN 9781605502977.

Gerard Schwarz. *Music basics*. Khan Academy, 2018. `https://www.khanacademy.org/humanities/music/music-basics2/notes-rhythm/a/glossary-of-musical-terms?modal=1`, Accessed: 2019-09-24.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to Sequence Learning with Neural Networks. `https://arxiv.org/abs/1409.3215`, 2014.

Poldi Zeitlin and David Goldberger. *Understanding Music Theory*. Omnibus Press, 2002. ISBN 9780711986718.

Appendices

A. Training data statistics

	Pauses	Chords[1]	Notes	Uniques	Durations	Spans
Max	383	129	153	70	47	65
Min	129	0	0	2	1	0
Average	314.2	39.9	29.9	10.7	1.0	17.6

Statistics for the drum phrases in the training data.

	Pauses	Chords	Notes	Uniques	Durations	Spans
Max	363	384	384	144	384	56
Min	0	0	0	1	1	0
Average	75.3	217.6	91.1	23.9	8.8	14.8

Statistics for the guitar phrases in the training data.

	Pauses	Chords	Notes	Uniques	Durations	Spans
Max	360	384	384	57	384	43
Min 0	0	0	0	1	1	0
Average	66.3	9.0	308.6	10.7	12.2	12.8

Statistics for the bass phrases in the training data.

	Pauses	Chords	Notes	Uniques	Durations	Spans
Max	333	384	384	113	384	67
Min	0	0	0	1	1	0
Average	49.0	262.9	71.2	14.7	215.9	12.4

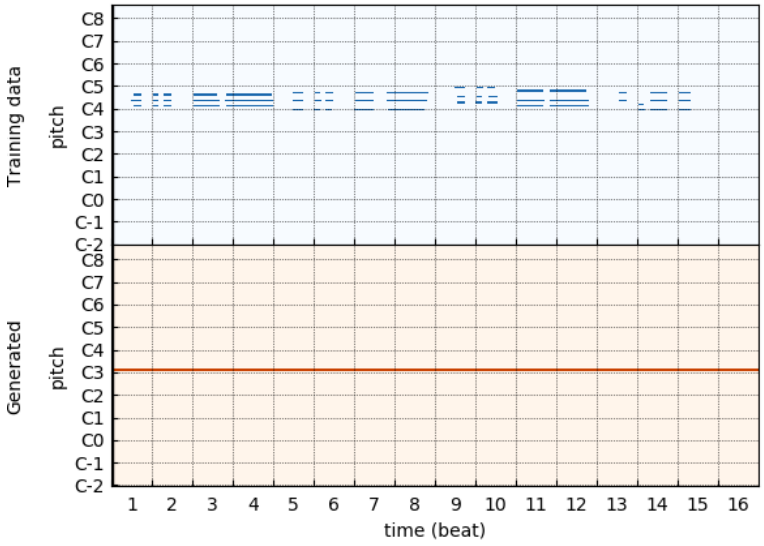
Statistics for the strings phrases in the training data.

[1]Chords in this context are defined as two or more drums being played at the same time, for example the kick drum and hi-hat

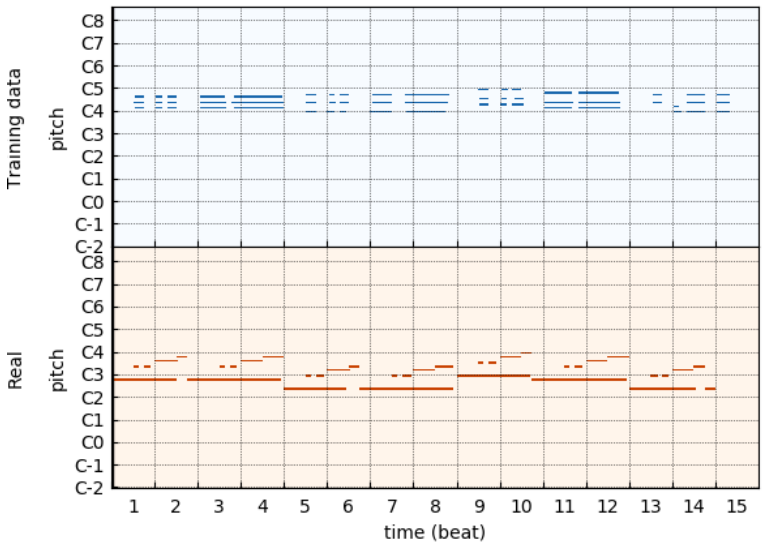
	Pauses	Chords	Notes	Uniques	Durations	Spans
Max	228	384	384	177	384	69
Min	0	0	0	1	1	0
Average	60.9	253.3	69.7	27.1	12.3	17.2

Statistics for the piano phrases in the training data.

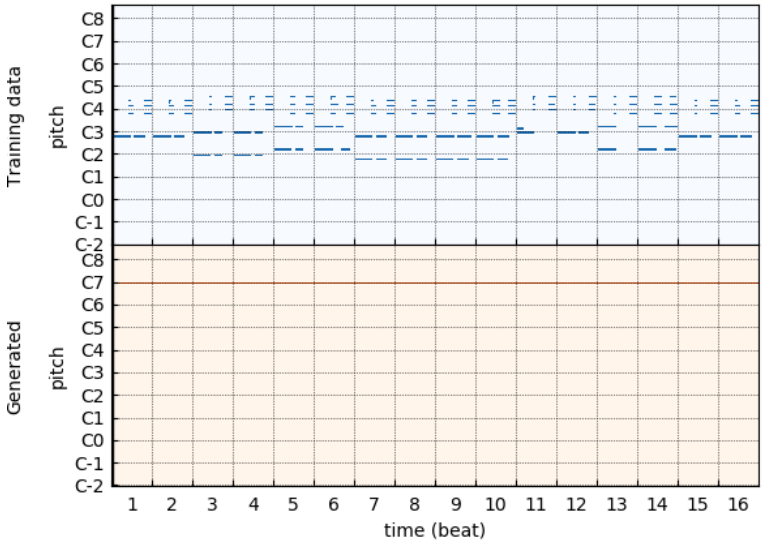
B. Generated and real samples



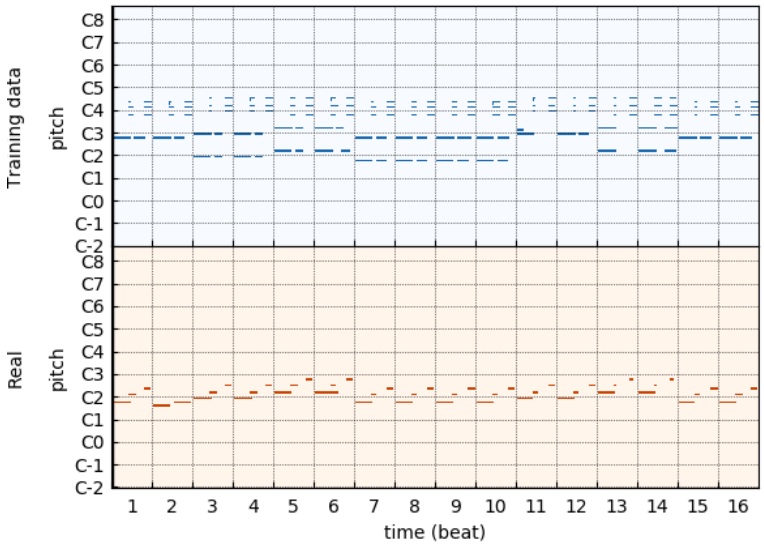
E1 – Generated accompaniment and melody.



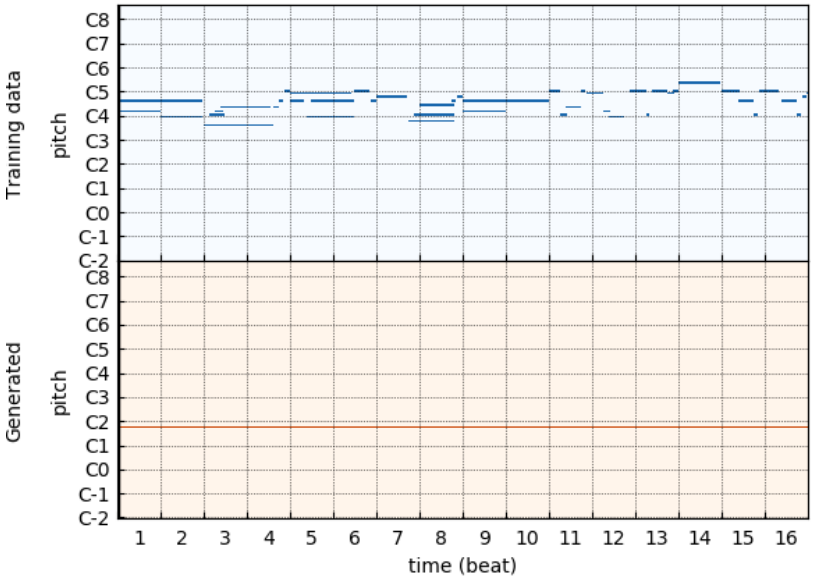
E1 – Real accompaniment and melody.



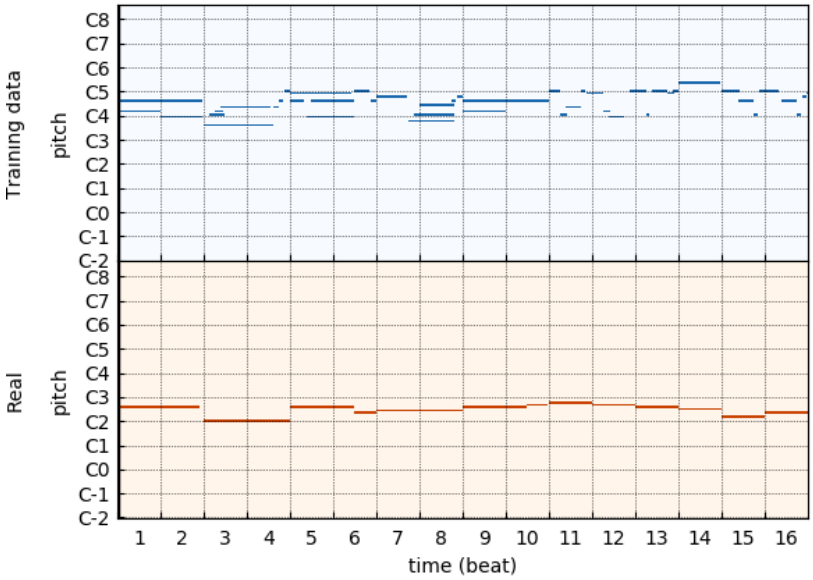
E2 – Generated accompaniment and melody.



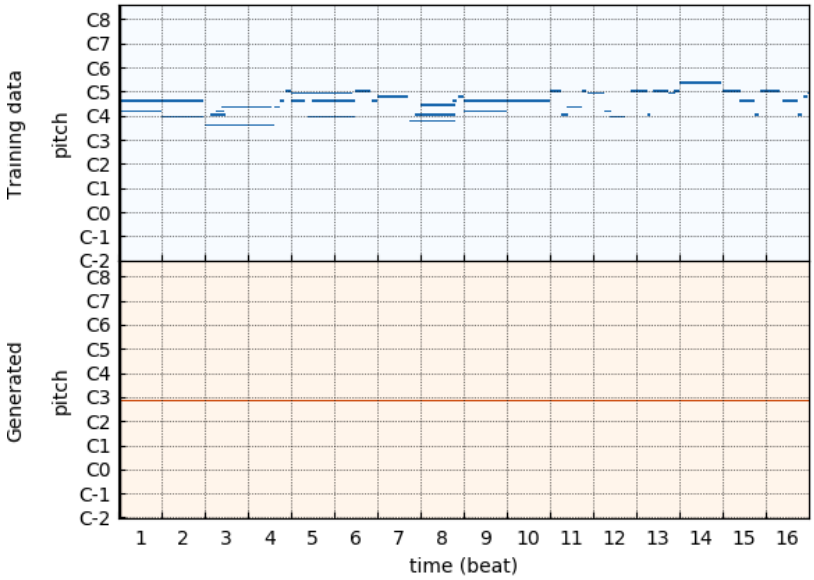
E2 – Real accompaniment and melody.



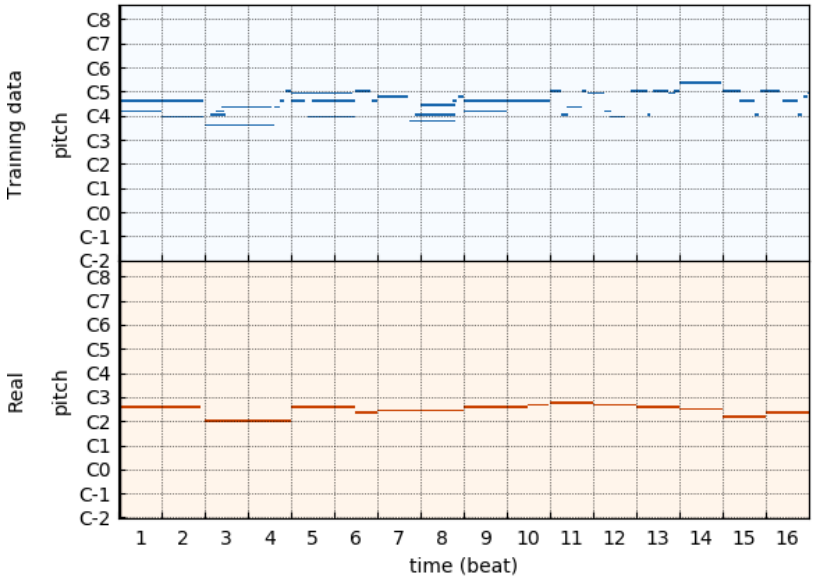
E3 – Generated accompaniment and melody with learning rate 0.01.



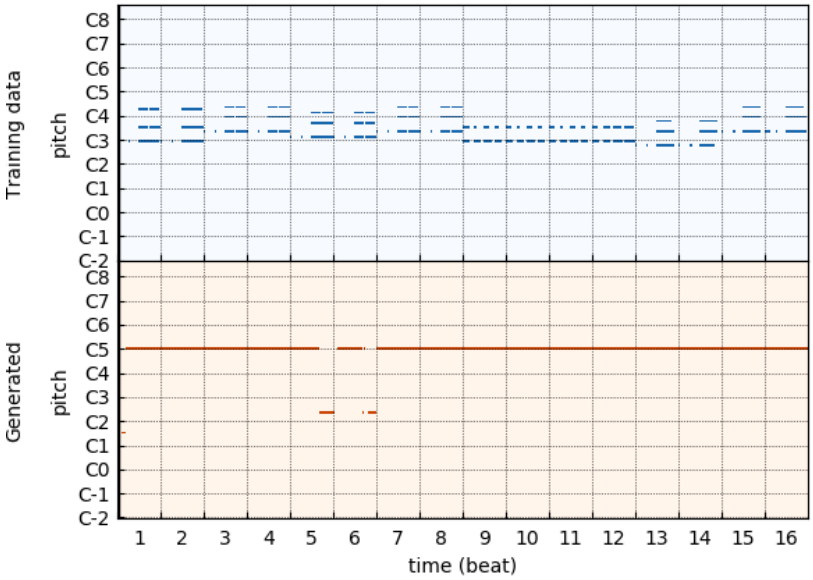
E3 – Real accompaniment and melody with learning rate 0.01.



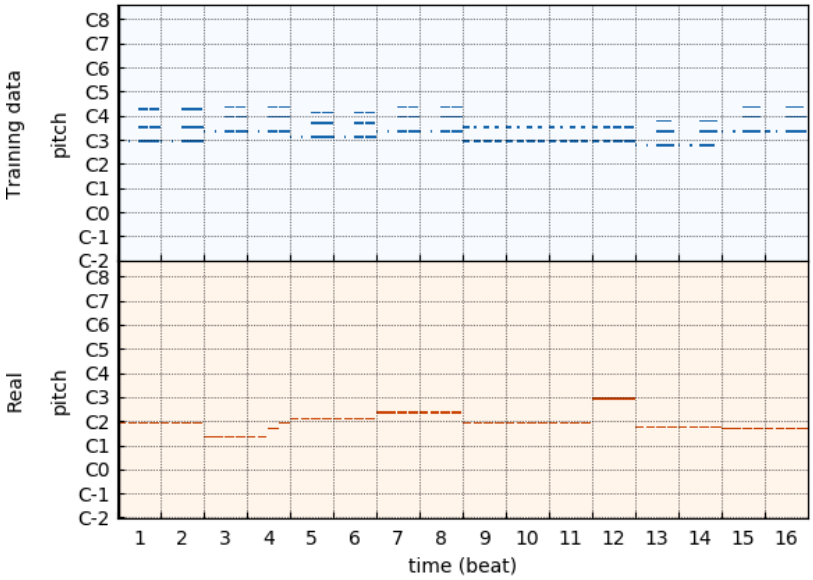
E3 – Generated accompaniment and melody with learning rate 0.005.



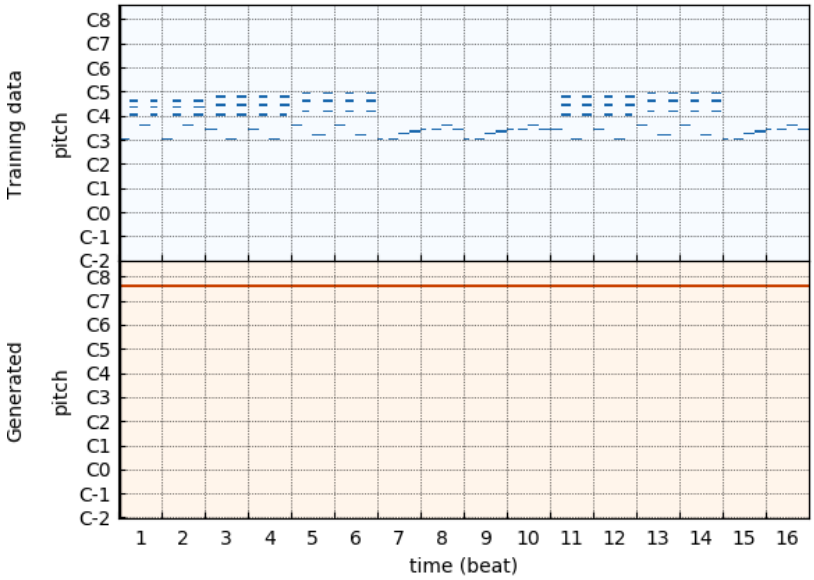
E3 – Real accompaniment and melody with learning rate 0.005.



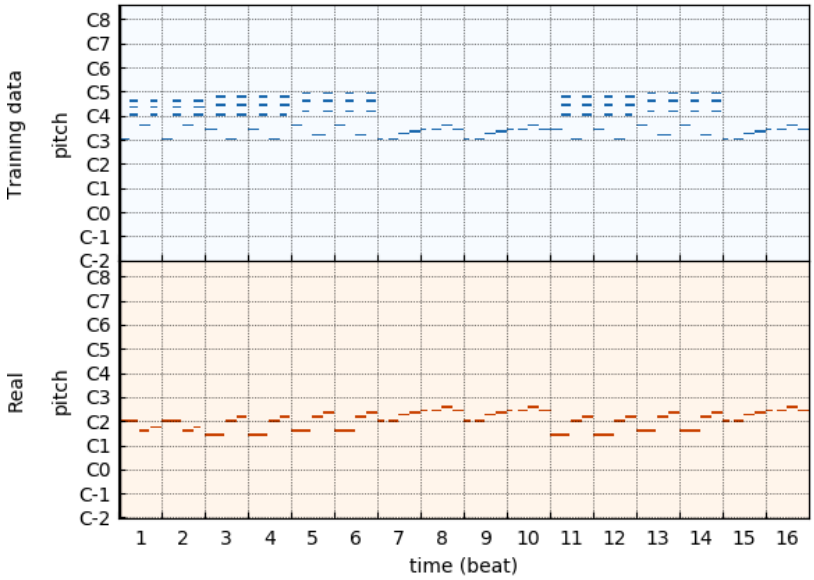
E3 – Generated accompaniment and melody with learning rate 0.001.



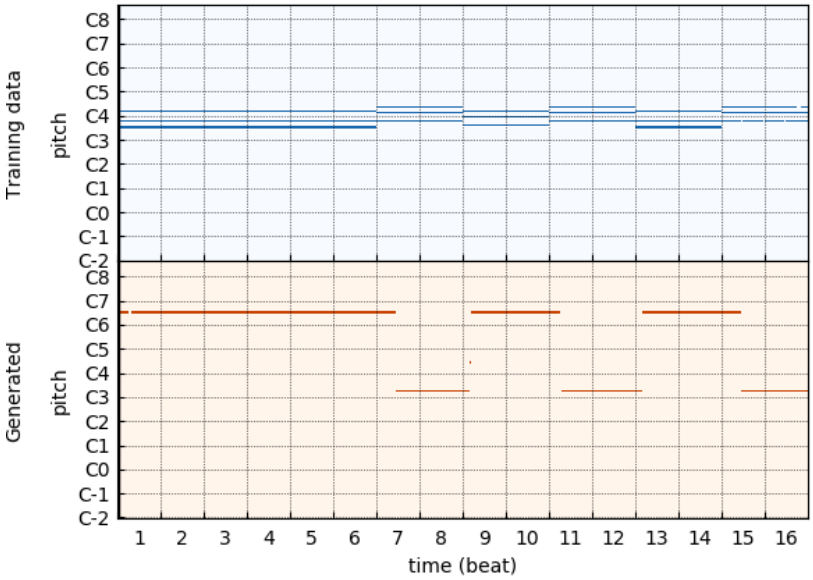
E3 – Real accompaniment and melody with learning rate 0.001.



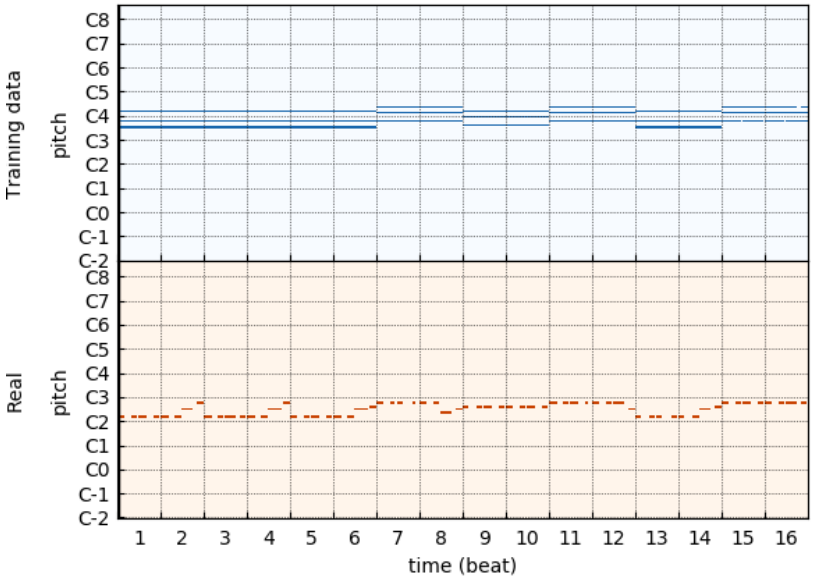
E4 – Generated accompaniment and melody with window size 6.



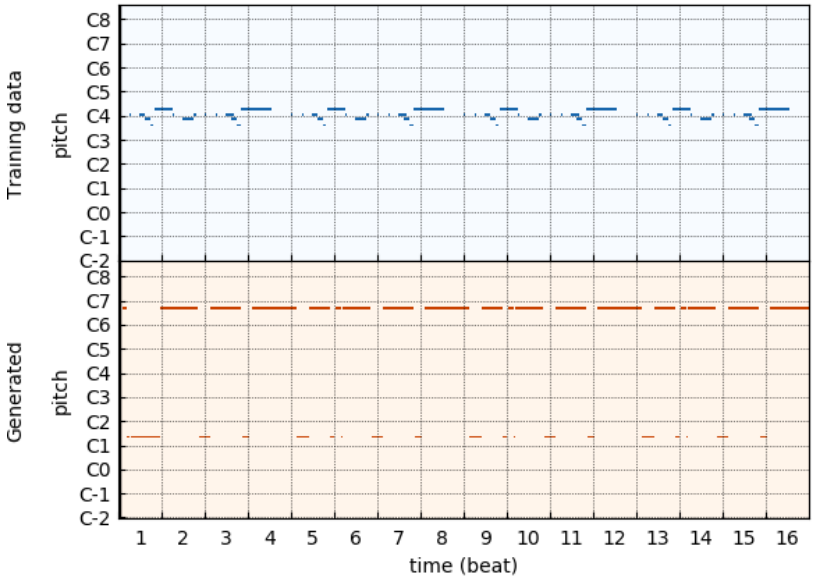
E4 – Real accompaniment and melody with window size 6.



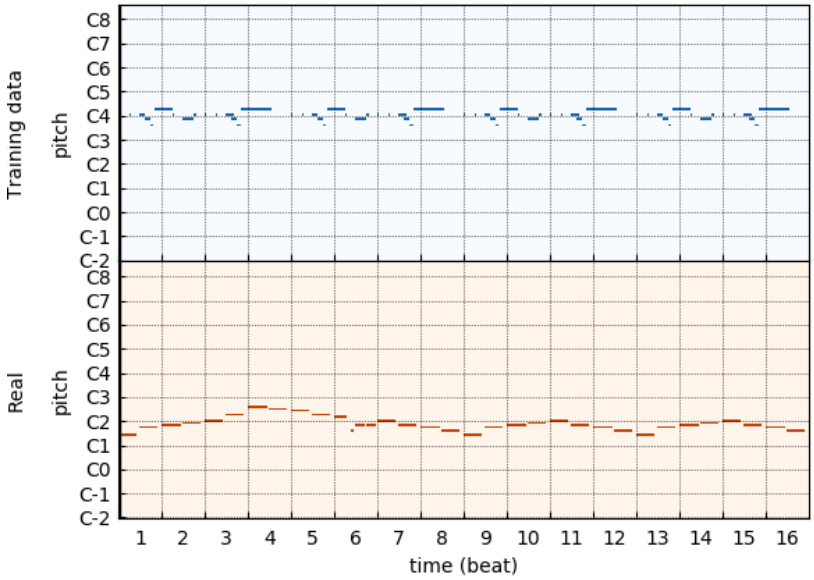
E4 – Generated accompaniment and melody with window size 12.



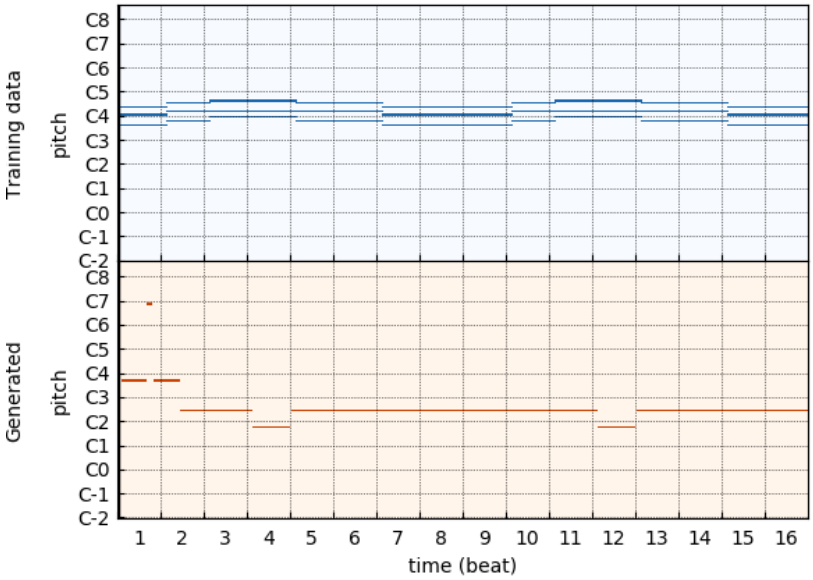
E4 – Real accompaniment and melody with window size 12.



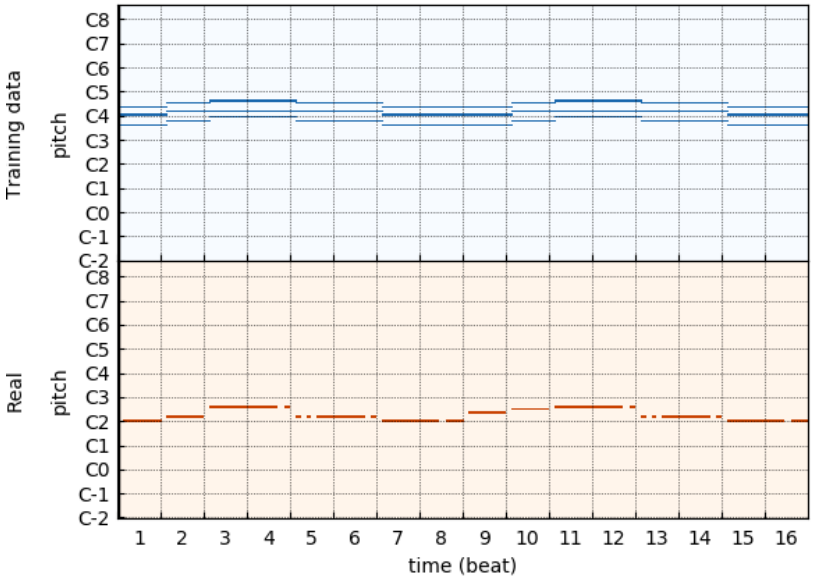
E4 – Generated accompaniment and melody with window size 24.



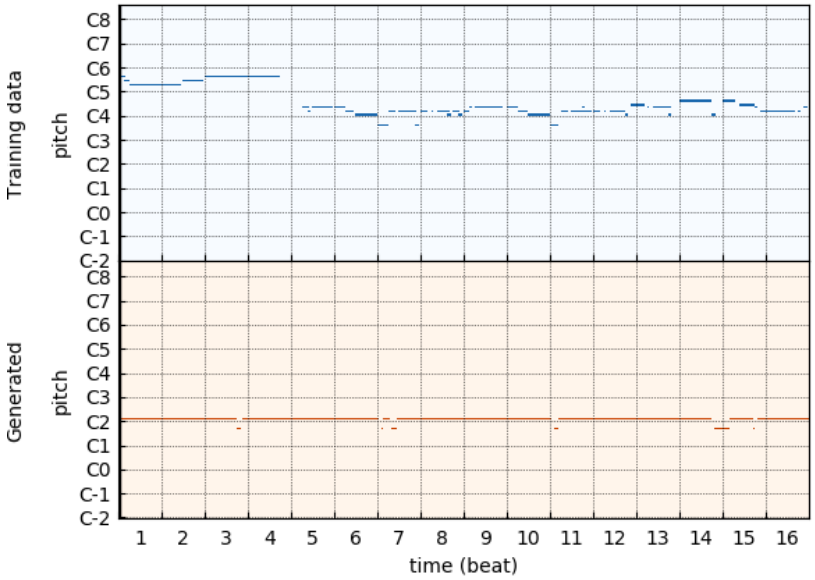
E4 – Real accompaniment and melody with window size 24.



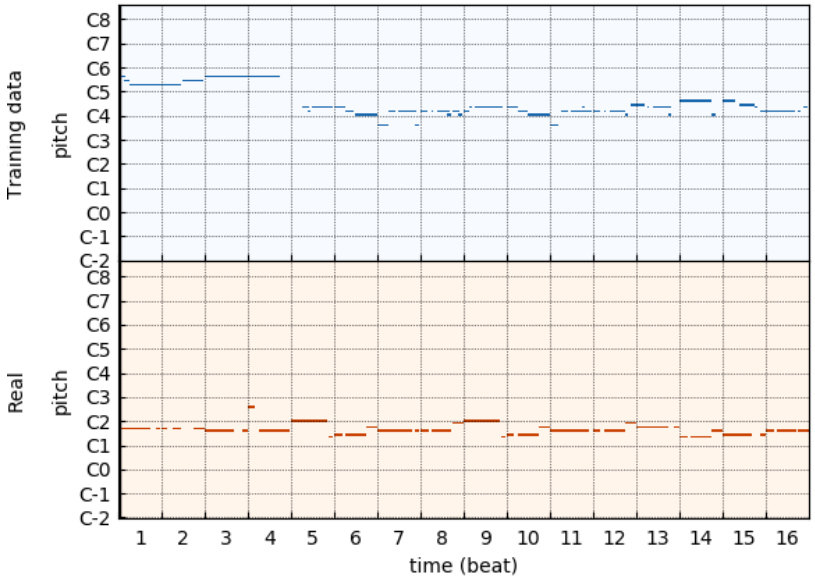
E4 – Generated accompaniment and melody with window size 48.



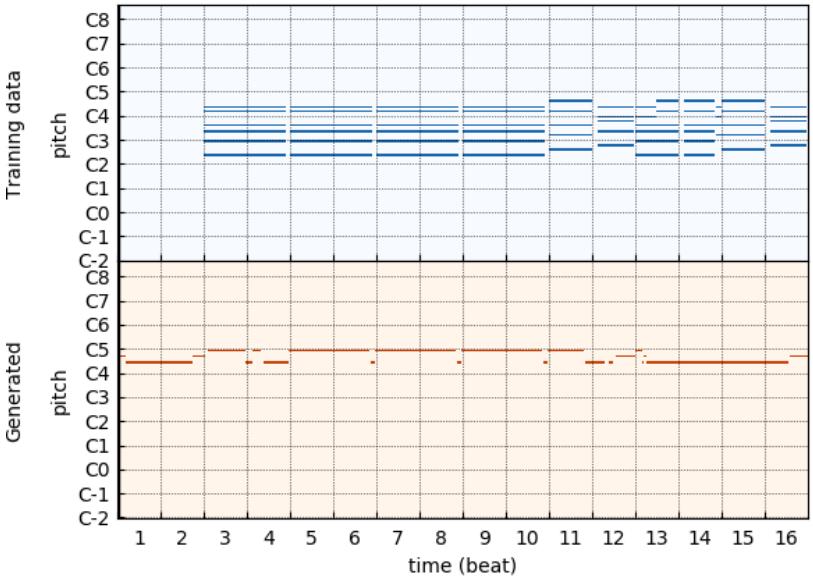
E4 – Real accompaniment and melody with window size 48.



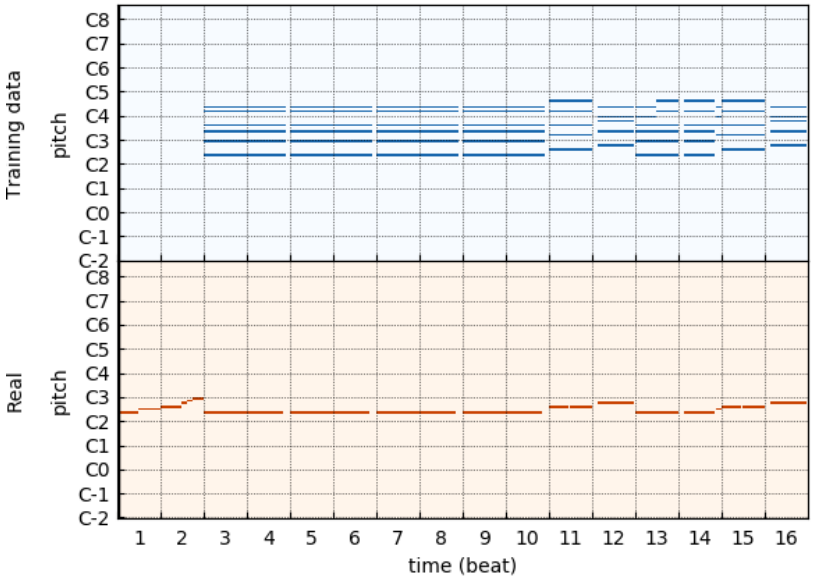
E5 – Generated accompaniment and melody with tonal range 24.



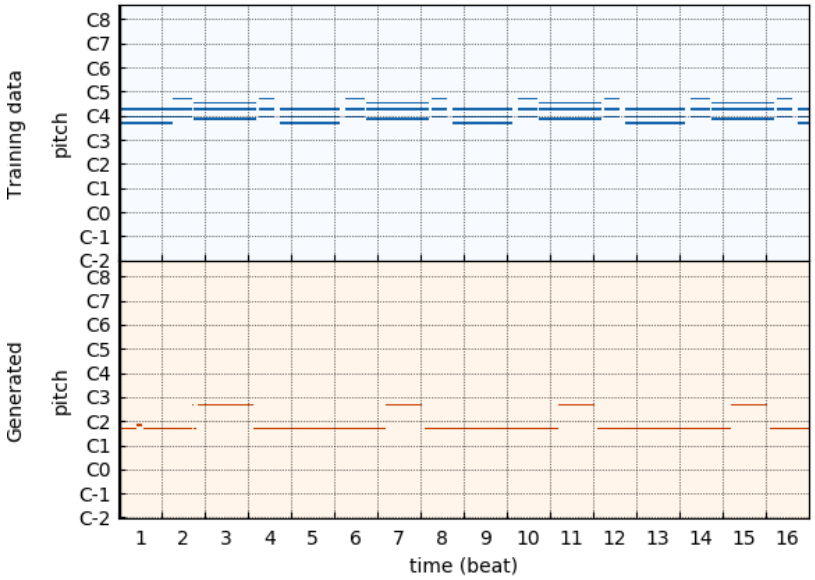
E5 – Real accompaniment and melody with tonal range 24.



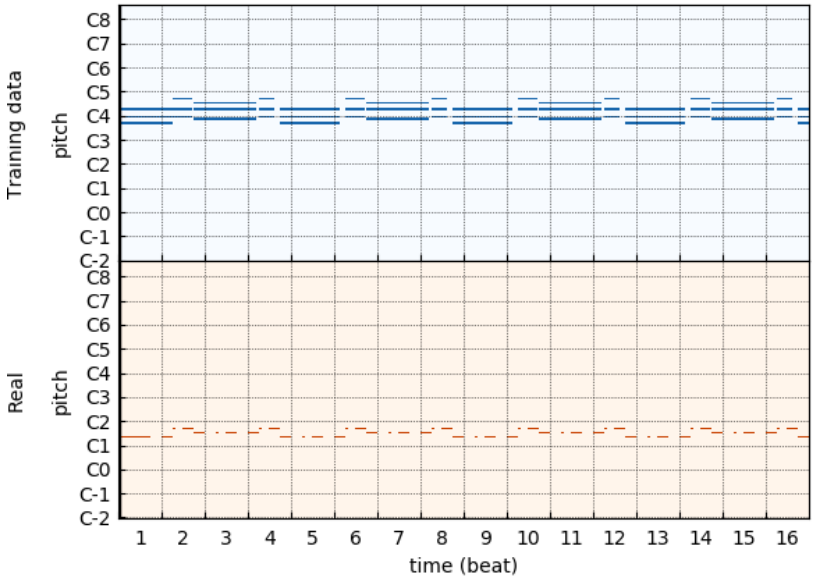
E6 – Generated accompaniment and melody with full tonal range.



E6 – Real accompaniment and melody with full tonal range.



E6 – Generated accompaniment and melody with reduced tonal range.



E6 – Real accompaniment and melody with reduced tonal range.

Endre Valstad Berg

Automatic Accompaniment Generation Using GANs

NTNU
Norwegian University of
Science and Technology