TECHNISCHE UNIVERSITÄT ILMENAU

Faculty of Computer Science and Automation

Department of Automation Engineering

Bachelor's Thesis

Generation of Symbolic Music Based on MusicVAE

Jakob Lerch

born September 10th, 2000 in Sangerhausen

Supervising professor:	Prof. Dr. Yuri Shardt
Industrial supervisors:	Dr. Andrew McLeod Dr. Jakob Abeßer

Degree program:	Ingenieurinformatik
Matriculation number:	61563
Date of submission:	July 3rd, 2023

Honor Statement

I, Jakob Lerch, hereby declare that I have written this thesis independently and have not used any other sources than those specified. All thoughts that have been taken either directly or indirectly from other sources have been explicitly identified as such. This thesis has not been submitted for credit in any other degree program or institution nor has it been previously published.

(Jakob Lerch)
Ilmenau, July 3rd, 2023

Eidesstattliche Erklärung

Hiermit versichere ich, Jakob Lerch, dass ich die vorliegende Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet. Diese Arbeit wurde in gleicher oder ähnlicher Form bisher noch keiner anderen Prüfungsbehörde vorgelegt oder veröffentlicht.

Abstract

Automatic **m**usic **g**eneration (AMG) systems are computer-based models that produce signals that can be interpreted as music, such as waveforms or note sequences. Apart from purely commercial motivations, like the creation of a potentially unlimited amount of music, such systems are also interesting from a creative point of view, since they could be used for support during the music composition process, or even be understood as a new type of instrument. In this work, the flat variant of the AMG system *MusicVAE* is re-implemented and trained. It is used for unconditioned generation of monophonic note sequences of two bars in length. MusicVAE is a variational auto-encoder. It maps a given note sequence to a multivariate Gaussian distribution in a lower-dimensional space (called *latent space*). From that distribution, it samples a vector and decodes it to another note sequence. The model can be trained so that the reconstructed sequence is as similar as possible to the input sequence. By randomly sampling and decoding vectors from the latent space after training, new pieces of music can be generated. In this work, a set of generated note sequences is compared to a held-out test set using selected objective criteria and a subjective analysis. It is shown that the trained model struggles to reproduce the rhythmic structure of the training data and that generated note sequences often contain large jumps in pitch and single non-diatonic notes. Despite such outlying notes, most note sequences contain coherent and diatonic melodies. In particular, interesting rhythms and melodies with multiple high, short notes followed by few longer notes are generated. In future work, ways to condition the latent space could be examined to find possibilities of controlling the generation process.

Zusammenfassung

Automatische **M**usik**g**enerierungssysteme (AMG) sind computerbasierte Modelle, welche Signale produzieren, die als Musik interpretiert werden können, zum Beispiel Schwingungsverläufe oder Notenfolgen. Abgesehen von ausschließlich kommerziellen Motivationen, wie der Kreierung einer potentiell unlimitierten Menge von Musik, sind solche Systeme auch unter dem Aspekt der Kreativität interessant, da sie für die Unterstützung des Kompositionsprozesses genutzt oder sogar als eine neue Art von Musikinstrument verstanden werden können. In dieser Arbeit wird die flache Variante des AMG-Systems *MusicVAE* neu implementiert und trainiert. Dieses wird für die unbedingte Generierung monophonischer Notenfolgen mit einer Länge von zwei Takten verwendet. MusicVAE ist ein Variations-Autoenkodierer. Es bildet eine gegebene Notenfolge auf eine multivariate Gauß-Verteilung in einem niedriger-dimensionalen Raum (genannt *latenter Raum*) ab. Von dieser Verteilung zieht es nach dem Zufallsprinzip einen Vektor und dekodiert diesen wieder zu einer Notenfolge. Das Modell kann so trainiert werden, dass die rekonstruierte Notenfolge so ähnlich zu der Eingabefolge ist wie möglich. Nach abgeschlossenem Training können neuartige Musikstücke generiert werden, indem Vektoren zufällig aus dem latenten Raum gezogen und dekodiert werden. In dieser Arbeit wird ein Satz von generierten Notensequenzen mit einem vorbehaltenen Testdatensatz verglichen, indem ausgewählte objektive Kriterien angewendet und eine subjektive Analyse durchgeführt werden. Es wird gezeigt, dass das trainierte Modell die rhythmische Struktur der Trainingsdaten nur mit Schwierigkeit reproduzieren kann und generierte Notensequenzen oft große Tonhöhensprünge und einzelne nichtdiatonische Noten aufweisen. Trotz solcher unpassenden Noten beinhalten die meisten Notensequenzen zusammenhängende und diatonische Melodien. Insbesondere werden interessante Rhythmen und Melodien mit mehreren hohen, kurzen Noten, gefolgt von wenigen längeren Noten generiert. Künftige Arbeit könnte Wege untersuchen, den latenten Raum zu konditionieren mit dem Ziel, Möglichkeiten zu finden, den Generationsprozess zu kontrollieren.

Acknowledgments

The author would like to thank Prof. Shardt, Dr. McLeod, and Dr. Abeßer for the close supervision throughout the whole writing procedure, and the Fraunhofer IDMT that he was allowed to use their resources for his work and for the opportunity of collaboration in general.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

Music generation is the task of computer-based creation of signals that can be interpreted as music. Systems with the purpose to fulfill this task are called ***a**utomatic **m**usic **g**eneration (AMG) systems*. Such systems could help finding new ideas during composition, be used as companions during live performances, create a potentially unlimited amount of music for creative productions, and they could even be understood as a new type of instrument, thus adding a new dimension to making music.

However, there are some ethical issues about creative content generated by machines. Training models on copyrighted data is a gray area (Creative Commons, 2021), so it is hard to say if it is even legal to use such models commercially. Furthermore, it can be discussed if it is even acceptable to use develop and deploy an AMG system for the creation of "new" art or commercial success, where the basis for it is actually the art of other people. Another potential drawback of AMG systems could be a decreasing demand for professional creators in the future. For example, it could be financially more favorable for a marketing team to buy a software for producing jingles than hiring a composer. In contrast, AMG systems can be used by artists themselves, serving as a supportive component in the creative process.

Music can either be generated directly as a waveform, as the systems proposed by Agostinelli *et al.* (2023) and van den Oord *et al.* (2016) do it, or it can generate a sequence of symbols, *e.g.* in the format of MIDI (Lehrmann, 1993), where a musical piece is just described, but can be interpreted. The second type will be called *symbolic music*, which is the subject of this thesis. Furthermore, music can be contain multiple notes played simultaneously, which is called *polyphonic music* or just one note at a time, not overlapping with other notes, which is called *monophonic* music. Monophonic music will be of interest in this work.

There are different tasks of symbolic music generation (see recent surveys by Briot (2021) and Hernandez-Olivan *et al.* (2022) for more details). They can be categorized by unconditioned and conditioned generation. Unconditioned generation is the task of generating new excerpts from scratch, *i.e.* without any prior or user-defined conditions. The style would be whatever the training data was. Conditioned generation is the task of generating a piece of music following given constraints, such as tempo

or key. Conditioned generation can again be divided into several subcategories: musical in-painting, style transfer, accompaniment, and continuation. Musical in-painting is the task of generating a sequence of notes that connect a starting and an ending piece of music (Huang *et al.*, 2017). Style transfer is the task of changing the style (which could mean *genre* or *composer* in the context of music generation) of a given piece (Brunner *et al.*, 2018). Accompaniment is the task of generating a musical voice dependent on other voices that are given. An example would be to generate a chord sequence to a given melody (Dong *et al.*, 2018). Continuation is the task of generating sequences following a priming sequence or note. Continuation and Accompaniment could also be considered as sub-types of conditioned generation (Eck & Schmidhuber, 2002).

The task considered in this thesis is generation of symbolic monophonic music where the focus will be on unconditioned generation and Western music. During the process of composition, music is often written down symbolically. A tool that generates music in that form could be useful in that case, rather than a waveform-generating system and, apart from that, such a tool can even be simpler, because it has a lower-dimensional output. A use case for unconditioned generation is the search for new impulses for the composition process. A system that is able to generate monophonic music can be used to compose leading voices or motifs. Using such a model, a composer could collect new impulses to look at his work from another perspective. Only Western music is considered, because of the musical background of the author.

The task has been worked on for many decades. An early approach to computer-based symbolic music generation was proposed by Hiller and Isaacson (1959). In their experiments, they generated pieces by choosing notes automatically following a set of rules. These rules were standard counterpoint composition rules as explained by Morris (1975) or formulated as Markov chains (for an explanation of Markov chains, see Gagniuc, 2017) with chosen transition probabilities. Today, many models use deep-learning approaches to model music, as it can be seen by the survey of Hernandez-Olivan *et al.* (2022). Due to the relevance of the approach of deep learning in the field of music generation, the focus here will be on recent and especially deep-learning methods.

In this thesis, the state of the art will be summarized. A model based on the AMG system *MusicVAE*, proposed by Roberts *et al.* (2018), will be implemented and trained. MusicVAE was chosen, because it is able to learn an embedding space that is shaped in a specific way and where each vector corresponds to a musical excerpt. This makes it possible to add conditioning in the future, for example by trying to influence the structure of the embedding space based on given distributions. The training dataset

will consist of monophonic note sequences of two bars, since that was also the case for the training data used by the authors of MusicVAE. Excerpts will then be generated and their quality be discussed.

Chapter 2: Musical Background

Cambridge University Press & Assessment (2023) defined music as "a pattern of sounds made by musical instruments, voices, or computers, or a combination of these, intended to give pleasure to people listening to it". The goal of this chapter is to introduce some fundamental concepts of music theory. For a more exhaustive introduction, the textbook by Gotham *et al.* (2021) can be consulted.

In Western culture, music is traditionally written using a specific notation called *score notation*. Two bars in score notation are shown in Figure 2.1. Part of that notation

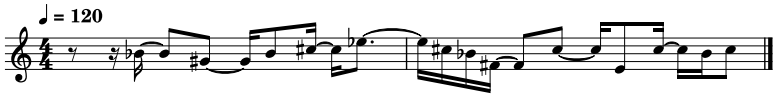


Figure 2.1: Two-bar musical score.

are symbols called *notes*. Each note corresponds to a tone being played or sung, and contains information about which fundamental frequency this tone has and how long it is to be held. A section of music without any notes is designated by one or more symbols called *rests*.

The set of possible pitches can be arranged over a discrete scale. The smallest unit of pitch that is considered in this thesis is called *semitone*. Twelve subsequent semitones form an octave. A pitch without the consideration of the octave is called *pitch class*. The pitch of a note is written as a combination of a pitch class and an octave. A pitch class corresponds to an element of the set $\{A,B,C,D,E,F,G\} \times \{\ ,\#\}$, where the symbol # raises the pitch class by one semitone. The specific octave considered is written as an index of the pitch class symbol where that index is in $[-1,9]$. As an example, A is a pitch class, but the corresponding specific pitches are A_{-1} to A_9. The lowest pitch class in an octave is C. Table 2.1 shows what notation corresponds to which semitone number within an octave.

Subsets of pitch classes that sound pleasant when played together can be defined. Such subsets are called *keys*. Each key has a pitch class that can be seen as its center. This pitch class is called *root*. Each pitch class can be the root of a key. A key is further described by the intervals in semitones between its root and the remaining pitch classes in the key, regardless of the specific root note. Two common types of keys are

Table 2.1: Pitch class names

Pitch class in semitones	Pitch class name
0	*C*
1	*C*♯
2	*D*
3	*D*♯
4	*E*
5	*F*
6	*F*♯
7	*G*
8	*G*♯
9	*A*
10	*A*♯
11	*B*

Table 2.2: Note value for note lengths expressed in 16^{th} notes.

Note value	Duration in 16^{th} notes
16^{th} note	1
eighth note	2
quarter note	4
half note	8
whole note	16

the major and the natural minor key (here, just *minor key*). The pitch classes contained in a key relative to the root note in the case of a major key are $0,2,4,5,7,9,11$ and those contained in a minor key are $0,2,3,5,7,8,10$. A key is then referred to as *e.g.* the *key of C minor*. An excerpt that generally belongs to one of these keys is said to be diatonic[1].

Different note lengths are defined in relation to each other. The smallest note length considered in this thesis is a 16^{th} note. All other note lengths (considered in this thesis) can be expressed in multiples of 16^{th} notes. Notes can be named based on their duration. This name is then called *note value*. Table 2.2 shows what note value corresponds to which number of 16^{th} notes. If a note with a length of l 16^{th} notes is dotted, *i.e.* if there is a dot right after it in the score notation, its duration becomes $l + l/2$. The point in time where a note is started to be played is called *onset* and the point in time where a note is stopped being played is called *offset*. Rhythm can be seen as the pattern of onsets and releases of notes in a piece. The sequence of notes in a

[1]The concept is complicated, but interested readers should refer to Kostka *et al.* (2018)

score can be subdivided into intervals called *measures* or *bars*. Each measure contains specific points in time to which the rhythm is oriented. These points are called *beats*, they are usually evenly spaced throughout a measure. In this thesis only pieces will be considered where every measure contains four beats and one beat consists of four 16^{th} notes. This pattern is referred to as a *time signature of ¼*.

The positions in a bar can be organized hierarchically into levels. Every first beat called a *downbeat* and therefore said to be on *downbeat level*. Every second beat is said to be on *half-note level*, every fourth, on *beat level* or *quarter-note level*. These levels are referred to as *metrical levels* in general and the specific positions (*e.g.* the 2^{nd} downbeat) are called *metrical positions*. There are also smaller metrical levels: a metrical position on the first or third 16^{th} note of a beat is said to be on *eighth-note level* and a metrical position on a 16^{th} note is said to be on *16^{th}-note level*.

In computer software, notes can be stored in different formats. One format is ***M**usical **I**nstrument **D**igital **I**nterface (MIDI)* (Lehrmann, 1993). Originally, it was a protocol for music instruments containing a digital interface communicating with each other. However, it can also be used to store musical notation as a file. The MIDI standard defines a pitch as an integer in $[0, 127]$ with a pitch of 60 corresponding to C_4. This convention will be adopted for this work.

One point to be considered in this thesis is the long-term structure of music. A musical piece is build up of multiple measures that together form a higher-level structure. This repeats over multiple levels. This can easily be seen by listening to an arbitrary pop song, where multiple bars are built together to form sequences of bars that again form a structure of verses, refrains, and interludes. Thus, the structure of music is hierarchical, which is important to note for subsequent chapters.

Chapter 3: State of the Art

Several mathematical models were used for symbolic music generation, such as Markov chains (Hiller & Isaacson, 1959) and neural networks (Todd, 1989). In this chapter, an insight into the state of the art of AMG is provided.

Many recent approaches can be found within the field of deep neural networks (Hernandez-Olivan *et al.*, 2022). A basic textbook providing an introduction to that field is the one by I. Goodfellow *et al.* (2016a). It can be consulted for a deeper insight into the topic. Basically, a neural network is a mathematical model containing many parameters. The examples named in this thesis can be seen as models to approximate an unknown function. To achieve that, the parameters of such a model have to be optimized to minimize a given loss function. For optimization, algorithms like stochastic gradient descent or variants and the back-propagation of the error signal are used (Bottou, 1999). For generation, a basic architecture is often used nested within another, higher-level model. The basic architecture will be called *backbone* and the higher-level model, *generative model*. The backbone is designed or chosen so that it can process the considered type of data, such as images or time series. The generative model defines the rough structure and the training procedure and makes use of the backbone to actually process the data.

Section 3.1: Backbone Architectures

Section 3.1.1: Fully-Connected Layers

One component that is used often in neural networks is the fully-connected layer (I. Goodfellow *et al.*, 2016a), which is an affine transformation followed by an activation function f:

$$y = f(Wx + b), \tag{3.1}$$

where f is the activation function, x is the input vector and can be of any dimension, and W is a matrix. W is called *weight matrix* and b, *bias vector*. All values of x, W, and b are in $\mathbb{R}$. The activation function can be any differentiable function. In the simplest case, the activation function is $f(x) = x$, but it can also be chosen to

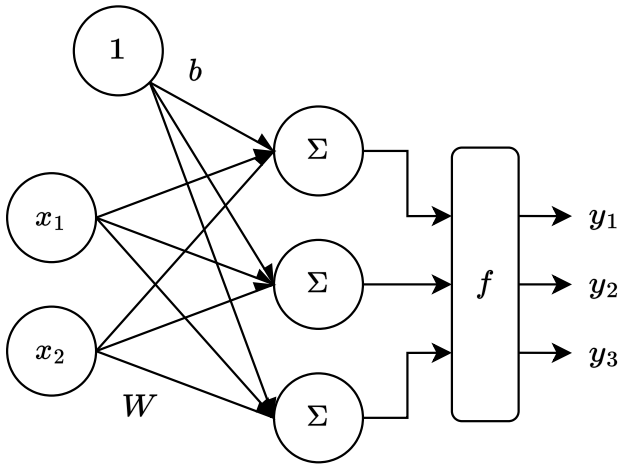


Figure 3.1: Fully-connected layer, where $x = (x_1, x_2)^T$ and $y = (y_1, y_2, y_3)^T$.

be non-linear, *e.g.* $f(x) = \tanh(x)$. When the term *fully-connected layer* is used in the following text without any comment on f, it is assumed that $f(x) = x$. A fully-connected layer therefore has an input dimension, which is equal to the dimension of x and an output dimension. In the following text, the output dimension will be called *dimension of the fully-connected layer*, since the input dimension is clear from the input. The fully-connected layer can also be seen as a directed non-cyclic graph, as visualized in Figure 3.1.

Section 3.1.2: Recurrent Neural Networks

Recurrent **n**eural **n**etworks (RNNs) are neural networks that contain feedback connections. They can be used to model sequential (*e.g.* time) data. However, the problem with traditional RNNs like those described by Todd (1989) is that during learning, the error that is propagated back through time either vanishes or increases exponentially. This is not desirable, since a very high gradient leads to the weights oscillating so the model does not find potential minima during training and a vanishing gradient leads to the model not being able to learn long-time dependencies.

Hochreiter and Schmidhuber (1997) therefore proposed **l**ong **s**hort-**t**erm **m**emory (LSTM), an architecture that enforces constant error flow. A state-of-the-art

Table 3.1: Names of the LSTM's components.

i_t ...	input gate	o_t ...	output gate
f_t ...	forget gate	c_t ...	cell state
g_t ...	input activation	h_t ...	hidden cell state

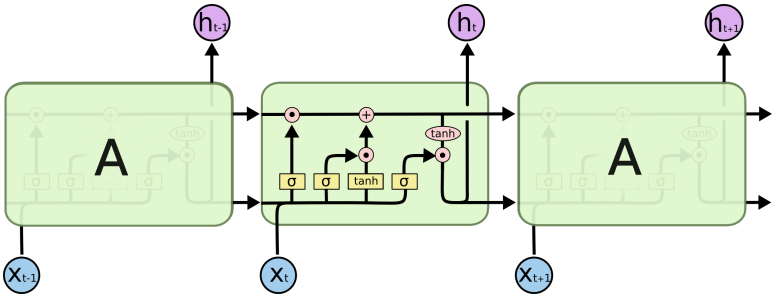


Figure 3.2: LSTM. Adapted with changes from Olah (2015).

LSTM can be described by the following equations (Sak *et al.*, 2014):

$$i_t = \text{sigmoid}(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \tag{3.2}$$

$$f_t = \text{sigmoid}(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \tag{3.3}$$

$$g_t = \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \tag{3.4}$$

$$o_t = \text{sigmoid}(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \tag{3.5}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \tag{3.6}$$

$$h_t = o_t \odot \tanh(c_t). \tag{3.7}$$

Variables with the index $t-n$ refer to the value of that variable to the $(t-n)^{\text{th}}$ time step. The symbol W refers to a weight matrix and the indices show what exact matrix is meant. The matrix W_{if} is the matrix that is applied to the **i**nput to yield a summand of the **f**orget gate. All other matrices and bias vectors are named following that scheme. The operation $\odot$ is the element-wise vector product, and $\text{sigmoid}(x)$ is defined as:

$$\text{sigmoid}(x) = \frac{1}{1+\exp(-x)}. \tag{3.8}$$

The concept of peephole connections, as mentioned by Sak *et al.* is omitted here. Each function is named following Table 3.1. The purpose of the input gate is to protect the state from irrelevant inputs. The output gate protects other layers from irrelevant values stored in the cell state (Hochreiter & Schmidhuber, 1997). The forget gate's purpose is to reset out-of-date cell state values. A graphic visualizing the equations above is shown in Figure 3.2. From Equations (3.2) to (3.7) and Figure 3.2, it can

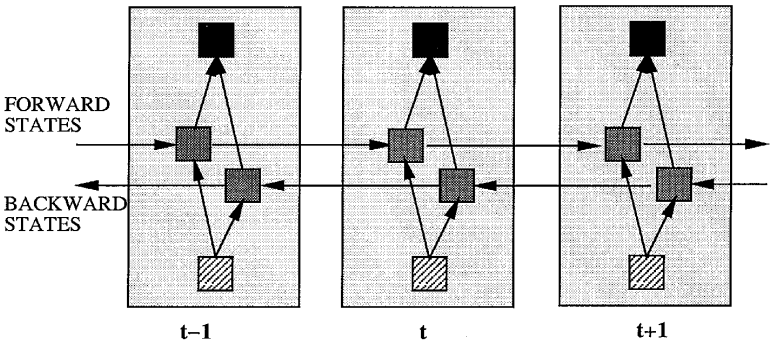


Figure 3.3: BRNN. Adapted from Schuster and Paliwal (1997).

be seen that the dimension of c_t and h_t has to be equal. The term *hidden-state size* therefore refers to both, the dimension of h_t and that of c_t. Given an input sequence of $(x_1, x_2, ..., x_T)$, where T refers to the number of time steps, the output of an LSTM is either the sequence $(h_1, h_2, ..., h_T)$ or just the final hidden state h_T.

There are several variants of LSTMs. Multiple LSTM layers can be stacked, in that sense that the input sequence to the next LSTM layer is the output sequence $((h_1, h_2, ..., h_T)$ of the previous LSTM layer). Schuster and Paliwal (1997) mention that it could be useful to incorporate future information into the current state or the output of an RNN. For this reason, they proposed **b**idirectional **RNN**s (BRNNs, or BLSTMs in case the RNN is an LSTM). They consist of two independent RNNs, the first processing the input data in positive and the second, in negative time direction simultaneously. A schematic is shown in Figure 3.3. The concept of multi-layer LSTMs and BLSTMs combined leads to an ambiguity: a BLSTM with multiple layers could either be two multi-layer LSTMs that are combined to one BLSTM or multiple BLSTMs stacked onto each other. To resolve this ambiguity, the former will be called *multi-layer BLSTM* and the latter will be referred to as *stacked BLSTM* from now on.

Section 3.1.3: Other Backbone Architectures

There are many other components that can be used. Since they are relevant for explaining example AMG systems in Section 3.4, convolutional neural networks and transformers will be introduced in this section.

Convolutional **n**eural **n**etworks (CNN) were originally proposed by Fukushima (1980). The input to a convolution layer can be a tensor, which is called *feature map* in this context. The convolution layer itself is defined by another tensor, called *kernel*. The convolution is then performed by sliding the kernel across the input feature map. For each kernel position the output of the convolution operation for that position is

calculated by multiplying the corresponding elements of kernel and input feature map and summing the resulting products. By sliding the kernel across the input feature map repeating this operation for every position the result is another tensor, called *output feature map*.

As it can be read in the survey by Hernandez-Olivan *et al.* (2022), much attention has recently been paid to the transformer architecture proposed by Vaswani *et al.* (2017). Transformers are designed to process sequential input data, but in contrast to RNNs, transformers can process a sequence all at once without requiring sequential computations per element of the input sequence. Instead, it is entirely based on a mechanism called *attention* to retrieve the context of a token. Huang *et al.* (2018) and Jiang *et al.* (2020) used a transformer for sequence generation.

Section 3.2: Generative Models

The purpose of a generative model is to generate samples from an underlying distribution (Bond-Taylor *et al.*, 2022). As Bond-Taylor *et al.* explained, There are several types of generative models. Apart from other models, the variational auto-encoder will be discussed in more depth, since it is relevant for understanding MusicVAE. One of the other generative models explained is called *autoregressive model*. It models the data to generate as a sequence and generates that sequence element by element and the probability of one element being generated depends on all elements in the sequence that were generated before (Y. Bengio *et al.*, 2003). Autoregressive generation can be applied using *e.g.* RNNs. The input for the first time step can be given. The output of that time step is then fed back as the input for the next time step. All output elements together form the generated sequence.

I. J. Goodfellow *et al.* (2014) proposed an architecture called ***g**enerative **a**dversarial **n**etwork (GAN)*. It consists of a generative model (called *generator*) and a discriminative model (called *discriminator*). The generator's task is to learn the distribution of the training excerpts, while the discriminator's task is to estimate the likelihood that the example is generated by the generator or drawn from the training set. During training, the generator's goal is to learn the distribution of the training set so well that the generated data cannot be distinguished from training data anymore, *i.e.* that the discriminators decision if a sample is generated or original is wrong has a probability of 0.5.

The generative model used by MusicVAE (Roberts *et al.*, 2018) is the **v**ariational **a**uto-**e**ncoder, proposed by Kingma and Welling (2014), so it will be ex-

plained in more detail. First, some notation has to be clarified. Kingma and Welling (2019) stated that in machine learning, one is often interested in finding probabilistic models of phenomena from data and that such models can be written as probability distributions. Let an uppercase letter, like X, be a random variable. The sample space of the random variable is denoted as the script version of the letter representing the variable, here $\mathcal{X}$. Each element of the sample space is a vector of values of values of all observed variables, flattened, and concatenated. Unless stated otherwise, a sample is denoted as the lowercase variant of letter representing the random variable, for example, x. The probability distribution of a random variable X is denoted as lowercase letter with the variable it belongs to in brackets, *e.g.* $p(X)$. The probability of a specific sample x being drawn from such a distribution is denoted as $p(x)$ (meaning $p(X = x)$). Let $p_\theta(X)$ be a probabilistic model that approximates the true, but unknown distribution of X. The variable θ denotes the parameters of that model. If the true distribution is conditional with respect to another random variable C, it can be approximated using a model $p_\theta(X|C)$. Following Kingma and Welling, conditioned variables can be considered as input to the model. The authors stated that one way to parameterize such a probabilistic model is by using neural networks. In the context of VAEs, the existence of another random variable Z (aside from X) is assumed. This variable is said to be unobserved and yet it is part of the model. Such a variable is called a *latent variable*.

Cinelli *et al.* (2021a) explained that a VAE consists of two probabilistic models, the *encoder* and the *decoder*. During forward propagation, a sample x gets fed into the encoder model that maps it to a distribution over the latent variable Z. This distribution is conditional with respect to X. Hence, the encoder is denoted as $q_\phi(Z|X)$ (with q being a probability distribution and ϕ being the parameters of it). The sample space of Z is called the *latent space* and an element in it is called a *latent vector*. Bowman *et al.* (2016) stated the distribution that x is mapped to is often chosen to be a Gaussian distribution with a diagonal covariance matrix. The encoder can be parameterized as a neural network that outputs the vectors $\mu(x)$ and $\sigma(x)$ of a multivariate Gaussian $\mathcal{N}(\mu(x), \text{diag}(\sigma(x)^2))$ where $\sigma(x)^2$ is the element-wise product of $\sigma(x)$ with itself (Kingma & Welling, 2019). From the resulting distribution, a sample z is drawn and passed through the decoder returning a distribution over X (Cinelli *et al.*, 2021a). This distribution is conditioned with respect to Z. Hence, the decoder is denoted as $p_\theta(X|Z)$. A sample from that distribution is denoted as $\hat{x}$. Cinelli *et al.* noted that the most probable sample from the output distribution should be similar to x after optimization. The overall structure of a VAE is shown in Figure 3.4.

Following Bowman *et al.* (2016), the optimization objective used for the training consists of two components. First, there is the reconstruction loss. The higher it

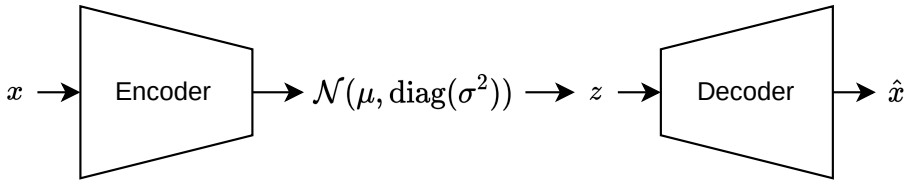
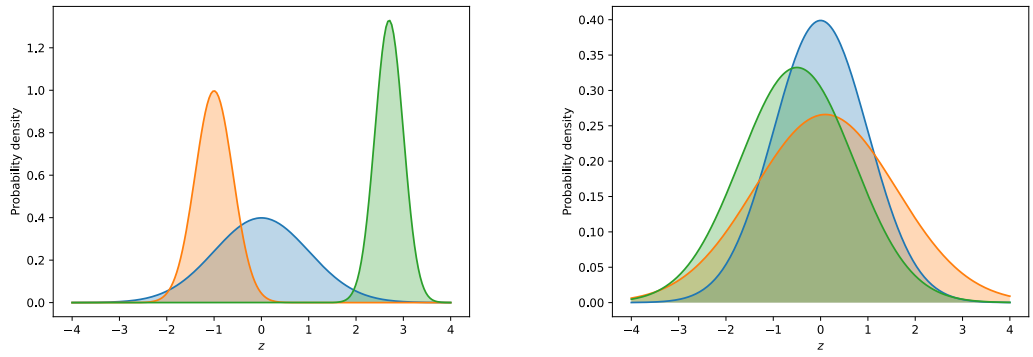


Figure 3.4: Variational auto-encoder.

is, the less similar $\hat{x}$ and x are. Second, there is the **K**ullback-**L**eibler (KL) divergence. The higher it is, the less similar the output distribution $q_\phi(Z|x)$ of the encoder is to a given distribution $p(Z)$. The latter is also called *prior distribution*. Bowman *et al.* noted that the model is regularized during training by the KL divergence term so that the output distribution of the encoder is close to the prior distribution, which is often a spherical Gaussian. They stated that this regularization leads to the decoder being able to reconstruct samples successfully when the given latent vector is sampled from the prior distribution. Another way to think about that is that the decoder learns to reconstruct only those latent vectors it gets as input. Those vectors are sampled from the output distributions of the encoder. If the output distributions of the encoder never have a high probability in a certain region, there probably would no latent vectors be drawn from that region and the decoder would never learn to reconstruct those. Such regions are called *holes*. Another effect of that regularization is that two latent vectors that have a small euclidean distance to each other are decoded to samples that are similar to each other (Cinelli *et al.*, 2021b). This can be verified by the fact that a sample x is encoded to a distribution. The optimization objective encourages the VAE to learn that a sample $\hat{x}$ that is decoded from a sample from that distribution should be similar to x. It also learns that the distributions of different inputs should overlap. Given two distributions are retrieved from the encoder and they are near to each other, maybe overlapping. In this case, the model learned to decode these neighboring sequences to highly different samples in the data space. If then a sample is drawn from the overlapping area, this sample is almost equally different to both of the different original data samples. This would lead to the optimization objective penalizing that behavior, hence similar data samples are likely to be encoded to neighboring regions in the latent space.

A latent space that has few holes and whose latent vectors which are near to each other decode to similar samples is considered desirable, hence it is called *well-formed*. Figure 3.5 shows two toy examples to illustrate in which case the latent space is well-formed and in which not. One reason why a well-formed latent space is desirable is that one could achieve unconditioned generation of new samples, which is the



(a) Not desirable.

(b) Desirable.

Figure 3.5: Visualization of two toy examples with a one-dimensional latent space to illustrate the meaning of *well-formed latent space*. It is assumed that a VAE was trained with two samples in the training dataset. The probability density functions of the corresponding encoder output distributions are shown as green and orange curves and that of the the prior distribution $p(Z) = \mathcal{N}(0,1)$ as a blue one. In case (a), the decoder probably did not get values of z within $[0.5, 1.5]$ as an input, which have a high probability under the prior distribution. In case (b), the decoder probably got most values as input that have also a high probability under the prior distribution.

subject of this thesis, by sampling a vector z from the prior distribution and passing it to the decoder. With a well-formed latent space and a model optimized to yield low reconstruction loss, the decoder would return an sample $\hat{x}$ that is likely to be in the distribution of the training data, *i.e.* in the case of music, $\hat{x}$ would be a reasonable-sounding musical piece in most cases.

The VAE algorithm contains a sample operation, namely the sampling of the latent vector. During training, the loss has to be propagated back through the model. To be able to apply standard back-propagation rules, Kingma and Welling (2019) suggest not to sample directly from a distribution $\mathcal{N}(\mu(x), \text{diag}(\sigma(x)^2)$, but to calculate the sample z in the following way

$$z = \mu(x) + \sigma(x) \odot \varepsilon, \tag{3.9}$$

where $\varepsilon \sim \mathcal{N}(0, I)$. Since only the computation path of μ and σ contain weights that have to be adjusted, the error does not have to be propagated through a sampling operation. This is called the *reparameterization trick*.

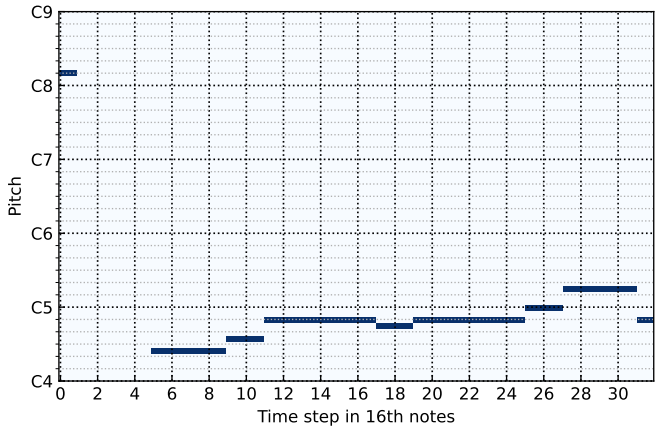


Figure 3.6: Piano roll.

Section 3.3: Representations Used for Symbolic Music Generation

When encoding a musical score, several aspects have to be considered: the onset of a note, its duration, the presence of rests, simultaneous notes, the presence of notes played subsequently without a rest. Each representation has to answer at least these questions.

Dong *et al.* (2018) and Yang *et al.* (2017) used piano rolls. In their context, a piano roll is an image of which note is played when, but notes are not written in the classical way. Instead, they are visualized in a diagram where the y-axis (discrete) corresponds to the pitch and the x-axis (discrete or at least theoretically continuous), to the time. A note is then displayed as a bar at a specific pitch, starting at a the onset time and having the length corresponding to the note's value. If there is a rest, there simply will be no bar at that time interval. Multiple notes played simultaneously are encoded as multiple parallel bars. The only difficulty is to represent two notes of the same pitch played after another, because in the most simple case that could be shown as one bar containing both subsequent notes. A piano roll is shown in Figure 3.6. Since Dong *et al.* and Yang *et al.* used CNNs, they can use these image representations directly for the processing with their models.

Other models encode music using token sequences, where each token encodes a time step with a specified length. In order for this to be possible, a given track has to be quantized. Given, the downbeat (first beat of a measure) times are known, this can be done by subdividing each measure by a specified fraction. Each token would then represent one fourth, one eighth, *etc.* of a subdivided measure. A problem

with quantization is that this could lead to the loss of information. As an example, there could be subsequent onsets of 16^{th} notes in a track. Such a sequence would get destroyed by a quantization by eighths. A policy of how the scores are quantized needs to be defined for such cases. It is also possible to quantize based on absolute time, instead of fraction of a measure, like Huang *et al.* (2018) did.

One token encodes an event occurring within its step, be it a note onset, a rest or the continuation of a note with a value longer than one time interval. A token could be *e.g.* a number, a string, or vector of numbers. Hadjeres *et al.* (2017) defined their tokens in the following way: onset is encoded with a token being the pitch and if a note is held for subsequent steps, the token has a specific "hold" value. In contrast, there is the representation used by Eck and Schmidhuber (2002): a note is represented by a hot-pitch vector (the indices of the current token vector correspond to the pitches of the played notes). They make no distinction between onset or continuation of a note. The lack of this distinction leads to the problem that if a sequence contains a sub-sequence of tokens having the same values, this could be interpreted as either multiple notes with the same pitch played after another or one long note. If this case is possible, it has to be defined how such a sub-sequence would be interpreted. Eck and Schmidhuber name alternatives to circumvent this issue, but stick to the first variant.

There is also the possibility to choose an encoding that depends on more features than just the absolute pitch. Mozer (1994), for example, did that. He encoded the music not per time step, but per note and chose to include the position on the circle of fifths and the position on the circle of pitch classes in the encoding in addition to the absolute pitch. The circle of pitch classes (also called *chromatic circle*) is a cyclic arrangement of notes where adjacent pitch classes are also adjacent on the circle. The circle of fifths is a cyclic arrangement of notes where adjacent pitch classes on the circle sound pleasing when played together. A similar encoding was chosen to encode the length of a note. The motivation of such a representation is that notes that are perceived as similar should have a small distance in the data space. As an example, C_0 and C_1 would have a smaller distance than C_0 and $F\sharp_0$. A visualization of the circle of fifths, the chromatic circle and the absolute pitch used as an encoding is shown in Figure 3.7

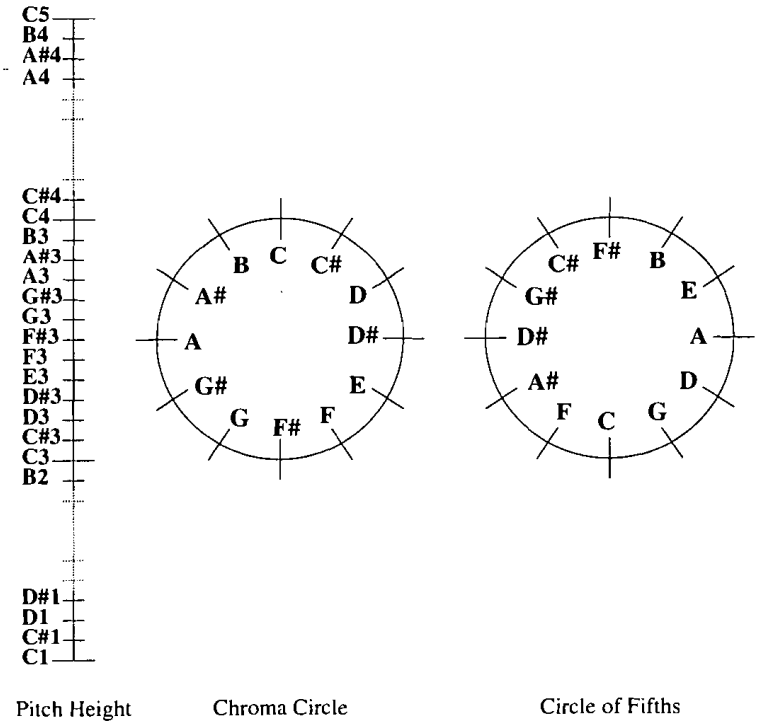


Figure 3.7: Representation using the absolute pitch (pitch height), the chromatic circle, and the circle of fifths. Adapted from Mozer (1994).

Section 3.4: Selected Examples of Music Generation Models

This section contains an overview of various approaches to unconditioned generation of symbolic music. Several models are compared to get an overview of the state of the art. For a more exhaustive recent overview see Hernandez-Olivan *et al.* (2022) and Briot (2021). Every model discussed in this section follows a specific generation scheme. What all have in common is that they have to be initialized somehow. There are different techniques to achieve this, *e.g.* setting a priming melody to be continued (that can also be sampled randomly to achieve unconditioned generation) or a vector with random values. Interesting is if the generated music sounds pleasant and if it shows long-term structure. However, these points are difficult to compare, because few works conducted the same evaluation procedures. Some models, *e.g.* the one proposed by Eck and Schmidhuber (2002) were only evaluated qualitatively by the author.

Early approaches used RNNs to generate a sequence autoregressively note by note. Todd (1989) described multiple architecture types that could be used for generation of note sequences. One of them is a three-layer network with two kinds of

input nodes: nodes containing recurrent connections both from the output nodes and themselves to store the context (called *context nodes*) and input nodes for initializing (called *plan nodes*). Here, the input vector to the plan nodes is called *plan vector*. Music is represented as a sequence of multi-hot vectors of size 15 containing one element for an "onset" signal. Every other element corresponds to a pitch in the key of *C*. One vector corresponds to a time slice with the length of a fourth note. The model is trained by learning a mapping from a set of plan vectors to a set of excerpts of 32 tokens. New excerpts are then generated by setting a new plan vector not being in the training set. A sequence is generated note by note feeding back the output. Due to the generated pitches being all in the same key, it is impossible for the model to generate pieces containing notes that might not fit to a specific key. This is a drawback since musical pieces throughout many different genres often do not stick to one key the whole time. More recent architectures do not make this assumption. Since the architectures proposed by Todd are simple RNNs, they also suffer from the vanishing or exploding gradient problem as it is explained in Section 3.1.2. The model did not learn rhythmical properties nor the relationships between pitch changes or rests and onsets well.

Eck and Schmidhuber (2002) encoded music as a sequence of multi-hot vectors, each having the duration of an eighth note. They subdivided the vector's elements into two sections: one that generates chords and another that generates melody. The network is trained to be initialized with a priming melody that is to be continued using an LSTM for autoregressive generation. The generated token sequences are of length 96, which is three times as long as the sequences generated by Todd. Following Eck and Schmidhuber, the generated excerpts sound like real music, but not with a good quality. They found that the model learns to incorporate long-term structure within the chord sequence, which is then used by the model to condition the melody. This suggests that a possible way to incorporate long-term structure could be the use of multiple components operating at different time steps.

Yang *et al.* (2017) proposed the model *MidiNet*. It is trained using a GAN and discriminator and generator both are CNNs. Music is generated sub-sequence by sub-sequence, where one sub-sequence consists of one bar, which is quantized by time steps with the length of one 16^{th} note each. A sequence is encoded as piano roll. A two-bar sequence is then generated by passing a random vector into the CNN to retrieve two bars of generated music. Apart from the generator and the discriminator, MidiNet contains a CNN called *conditioner CNN* that to encode a given piano roll to a lower dimensional representation, serving as conditioning information for the generator in addition to the random vector. This is used to generate multiple bars subsequently by

passing the last bar into the conditioner CNN to retrieve conditioning information for the current bar. Furthermore, the authors proposed an alternative that can be conditioned on a chord sequence that the melody in the generated sequence should follow. A listening test shows that MidiNet produces pleasant-sounding excerpts at the same level as similar models based on RNNs. Yang *et al.* pointed out that the first variant of MidiNet generates bars that sound unexpected, which might be due to the fact that the only information that is passed between bars for a longer time is the last bar. The chord-conditioned variant, however, performed better in the listening tests. The reason could be that the chords impose a long-term structure the melody then follows, just as with the model proposed by Eck and Schmidhuber (2002).

Roberts *et al.* (2018) proposed the architecture *MusicVAE*. It is a variational auto-encoder where both encoder and decoder consist of LSTMs. After training, new excerpts can be generated by sampling a single latent vector from the prior distribution and using it to retrieve the initial state of the decoder. The decoder consists of two LSTMs being arranged hierarchically. Thus, other than with the previous models discussed, hierarchy is incorporated into the design of the model and not achieved by the simultaneous generation or conditioning of chords. The first one decodes the latent vector to a sequence of vectors autoregressively, each representing one bar. Each of those vectors is then decoded by the second LSTM to generate bars. An excerpt is represented as a sequence of multi-hot vectors each representing a time step with the length of a 16^{th} note. The authors found that the hierarchical structure was able to copy the long-term structure of music. A user study suggested that there is no significant gap between the musical quality of MusicVAE and that of human-composed music. This model is interesting in particular, because of its hierarchical structure, seemingly good results and the fact that one excerpt can be generated easily by just sampling a single latent-vector from a probability distribution.

Another VAE, called *TransformerVAE* was proposed by Jiang *et al.* (2020), but the encoder and decoder do not consist of LSTMs, but of Transformers. Music is represented as a sequence of multi-hot vectors with the length of a 16^{th} note. Durnig training, excerpts are first encoded bar by bar leading to a sequence of latent vectors, each representing one bar. Then, this sequence is reconstructed to lead to the original excerpt. The latent vectors each contain representations based on the other vectors in the sequence. If the last bar is equal to the first one, the corresponding latent vector might contain the information “this bar looks like bar 1”. The model can then be used to encode a given sequence and change the properties of the whole piece by just changing a singe latent vector. The authors stated that the transformer model is another way of modeling long-term structure. TransformerVAE showed similar reconstruction

accuracies than variants of MusicVAE, hence this architecture could be promising to investigate further for music generation. To generate music unconditioned, the model could be adjusted to encode the latent vector sequence in a one-dimensional representation so that sampling of the latent vectors is easily possible, as in the case of MusicVAE.

However, there are other approaches to AMG than those based on neural networks, for example the work of Collins and Laney (2017). They combine Markov models (Gagniuc, 2017) with other processes, *e.g.* a pattern detection algorithm to ensure that the generated excerpts show long-term repetitive and phrasal structure. Yin *et al.* (2023) discussed that a disadvantage of rule-based systems is their high cost when it comes to designing them, but their reasons of their actions can be understood easily. Deep learning models, however, tend to learn weight configurations that are non-interpretable and need large datasets for training. They noted that the comparison of music generation systems tends to focus on other deep learning models, omitting non-deep-learning alternatives. The authors found that deep-learning methods did not outperform non-deep-learning methods and that there is still a lot of potential for improvement for AMG systems in general compared to human-composed music.

Section 3.5: Common Evaluation Procedures

One class of questions that is desirable to be answered is that of the more subjective ones, such as “Does the generated music sound pleasant?” or “Does the generated music sound artificially created?”. To evaluate those, subjective evaluation procedures, (*i.e.* listening studies) are performed, as it can be seen in the work of Dong *et al.* (2018), Roberts *et al.* (2018), and Yang *et al.* (2017). Following Yang and Lerch (2020), subjective evaluation should even be first choice to evaluate criteria that have to do with creativity. However, to conduct a user study well, resources, such as money, time, or subjects are needed and a lack of those can result in problems regarding reliability, validity and replicability of the results. Alternatively, generated excerpts could be evaluated qualitatively by the author. In this case, the goal would be to show that the generated music somehow looks like real music. Of course this does not replace a proper user study, because of the lack of a large ensemble of opinions creating a differentiated picture of the perception of the generated music, but it could be a useful part of the evaluation for this thesis.

Additionally, there are measures to evaluate music generation from an objective perspective. These can then be used for *e.g.* the analysis of a dataset of excerpts or the

comparison of different datasets like a test set and a set of generated music. Yang and Lerch (2020) proposed a complete framework to evaluate systems that generate symbolic music and, as a part of this, they look at possible objective evaluation measures. At first, the authors name the possibility to retrieve the likelihood of the generated data as it is done in other works, but they also mention that multiple aspects of the output should be evaluated as well. They proposed domain specific features that each can be calculated for a sequence of notes to retrieve a more interpretable representation of the data in terms of rhythm and pitch and could lead to a better comparability between systems if standardized:

1. Pitch count: number of pitches in a sequence
2. Pitch class vector: vector of which pitch class occurs how often
3. Pitch class transition matrix: a transition matrix computed by counting the pitch class transitions for each subsequent pair of notes
4. Pitch range: the difference between the highest and the lowest pitch in semitones
5. Average pitch interval of two subsequent pitches in semitones
6. Average inter-onset-interval: average time between two subsequent note onsets
7. Note length vector: which note length occurs how often?
8. Note length transition matrix: calculated in the same way as the pitch class transition matrix but instead of the pitch class the note length is chosen for calculation.

The term *pitch class* refers to the pitch modulo octave size. These features can then be used to retrieve insights to a single dataset or to compare multiple datasets. The distributions, mean, and standard deviation of the proposed features can be calculated and used to analyze a dataset or to compare multiple datasets using distance metrics for **p**robability **d**ensity **f**unctions (PDFs), *e.g.* the KL divergence, as proposed by the authors.

Apart from questions regarding the musical quality of generated excerpts, another important point to consider is the originality of the generated sequences, *i.e.* if generated excerpts were just learned to be copied from the training set or if they are truly new pieces. Some examples of what is possible in the context of originality evaluation are provided by Yin *et al.* They define measures that map two excerpts to $[0, 1]$ according to their similarity. One possible similarity measure is the cardinality score,

which is based on the intersection of the two pieces to be compared, where each piece is represented as a set of points. An alternative is the fingerprint measure, which is computed by hashing triples of notes and perform calculations based on the retrieved hashes. Apart from the use case of measuring the originality of one set in respect to another one, they also propose methods to measure the originality of the output of a music generation algorithm. They even come up with a procedure to incorporate originality measurement into the model itself.

Chapter 4: MusicVAE

MusicVAE was developed by Roberts *et al.* (2018) as a part of Google's Magenta project (Magenta developers, 2018). The general structure of MusicVAE is a variational auto-encoder with LSTMs as encoder and decoder each. Since the authors addressed several use cases with their model: generating two-bar sequences, 16-bar sequences, drum patterns, monophonic melodies, and triple-instrument tracks (trios). Hence, different variants were proposed. Since the main focus of this thesis is the unconditioned generation of monophonic symbolic music sequences, the generation of trios and drum patterns will not be considered, so that only long and short sequences of monophonic melodies remain. For generating short sequences, the authors created a baseline model containing a non-hierarchical decoder (called *flat model*). For generating long sequences, they created a model containing a hierarchical decoder (called *hierarchical model*). They trained and evaluated the baseline model for short sequences as well as for long sequences. The hierarchical model was only trained and evaluated for long sequences.

Section 4.1: Data

As a dataset, they collected and pre-processed MIDI data from the web. They conducted experiments with the **L**akh **M**IDI **D**ataset (LMD) (Raffel, 2016). For preprocessing, they removed all tracks that had no ¼ time signature, determined the bar boundaries using the encoded tempo, and quantized the tracks to 16^{th}-notes. This led to a set of tracks of arbitrary length. To retrieve sequences of a specific length, they extracted those sequences using a sliding window of two bars (16 bars) with a stride of one bar. They only added those sequences to their monophonic datasets where maximum one note was played at any time and contained one bar of consecutive rests at most. One element of such a dataset is called *excerpt*. Since they distinguish between drum and non-drum sequences, it is implicitly clear that they also sorted out drum tracks for their melody-sequence dataset. Each excerpt was encoded as a sequence of events. Each note was encoded as a one-hot vector of length 130. Possible token values are one of 128 onset values (one for each possible pitch), "offset", and "rest". Sets

of two-bar sequences and 16-bar sequences were created. They retrieved 28 million two-bar and 19.5 million 16-bar excerpts.

Section 4.2: Architecture

The encoder and decoder of MusicVAE are both LSTM networks that follow the structure proposed by Gers *et al.* (1999).

In every case, the encoder is basically a BLSTM that takes a sequence of vectors as an input to produce a vector of final hidden state. Its initial hidden and cell states are zero vectors. The encoder of the hierarchical model consists of two BLSTMs stacked onto each other[1]. Each BLSTM has a hidden-state size of 2048 per direction. The flat model is a single-layer BLSTM with a hidden-state size of 2048 per direction [2]. The forward and the backward hidden-state vectors are concatenated to get a vector h_T of size 4056. This is the input of two independent fully-connected layers with dimension 512 whose outputs are the vectors $\mu(x)$ and $\sigma(x)$ that parameterize the latent distribution $\mathcal{N}(\mu(x), \text{diag}(\sigma(x)^2))$. The activation function of the layer that produces $\sigma(x)$ is softplus(x) where

$$\text{softplus}(x) = \log(1 + \exp(x)). \tag{4.1}$$

Afterwards, the latent vector z is sampled from that distribution by using the reparameterization trick.

The decoder of the baseline model is a three-layer LSTM with a hidden-state size of 2048. Its initial state is retrieved by feeding the latent vector z through a fully-connected layer with a dimension of 12288 and $\tanh(x)$ as activation function (Magenta developers, 2017a, l. 80). The dimension of the resulting output vector is chosen so that it can be split into the initial hidden state and cell state vectors of each layer of the LSTM. The input to the LSTM at the zeroth time step is an initial token concatenated with z . To retrieve the output of the decoder, the hidden state of the last LSTM

[1] The authors stated that they use a "two-layer bidirectional LSTM network" with a hidden-state size of 2048 per direction as an encoder in the non-baseline model. Looking at the code, it becomes clear that what they mean is what we name stacked BLSTM networks (Magenta developers, 2017a, ll. 364–386). In the code, nonetheless, the configuration with stacked BLSTM networks can only be found in combination with longer sequences. Hence, the stacked BLSTM network is only used in the non-baseline model.

[2] In the official notebook (Magenta developers, 2017b), it is stated that for working with short sequences, a single-layer BLSTM with a hidden-state size of 2048 per direction was used. This suggests that the baseline model is the configuration found at Magenta developers (2017a, ll. 78–104). The model with the hierarchical decoder and the stacked BLSTM can then only be the configuration Magenta developers (2017a, ll. 364–386).

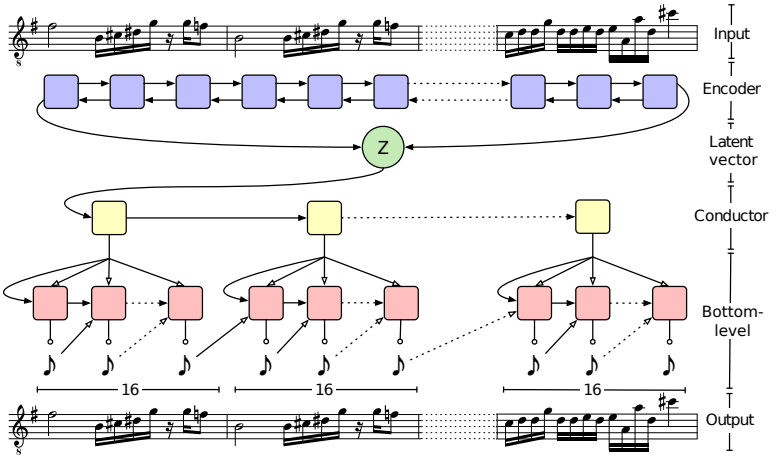


Figure 4.1: General structure of the hierarchical variant of MusicVAE. Adapted with changes from Roberts *et al.* (2018).

layer for every time step is fed through a fully connected layer and softmax(x) as an activation function, so each output token is basically a categorical distribution over the possible pitches. During generation, the resulting output of a time step is sampled from that distribution, concatenated with z, and fed back as input to generate the output of the next time step.

The hierarchical decoder consists of two subsequent LSTM networks. The first one is called *conductor RNN*. It generates a sequence of U embedding vectors from the latent vector, each representing a sub-sequence of the sequence to be decoded. In their model, they used a value of $U = 16$, so each generated embedding vector corresponds to a single bar[3]. The conductor RNN is in their case a two-layer LSTM with a hidden-state size of 1024 per layer. The initial state of the conductor RNN is retrieved by passing z through a fully-connected layer with a dimension of 4028, and $\tanh(x)$ as an activation function (just as in the flat decoder explained above). Afterwards, the embedding vectors generated by the conductor RNN are each fed into a shared fully-connected layer with tanh-function and dimension 512, thus producing the initial state vectors for the second network, which is called *bottom-level decoder RNN*. The bottom-level decoder RNN is a two-layer LSTM with a hidden-state size of 1024 per layer. It produces a sequence of output distributions, just like the flat decoder. The difference is that it does that for each embedding vector, leading to multiple sub-sequences. Analogous to the flat decoder, the output token of the bottom-level RNN of a time step is concatenated with the embedding vector of the current bar and then fed back as input for the next time step. As a result, there are U sub-sequences that

[3]The hierarchical model was evaluated only with 16-bar sequences.

are then concatenated to be the final reconstructed sequence. A schematic showing the hierarchical model can be seen in Figure 4.1.

One issue in the context of VAEs is that there is the possibility that the decoder does not learn to incorporate the latent vector into the decoding process. The authors refer to this phenomenon as *posterior collapse problem*. They limit the scope of the hierarchical decoder to make the model learn to use the latent vector for modeling long term structure. For this reason, the bottom-level decoder RNN was designed not to pass its state on over multiple sub-sequences, but to use the corresponding embedding vector to calculate the initial state of a bar. Hence, the only information that is passed from one bar to another is the last output token of the previous bar.

Section 4.3: Training and Outcomes

Two variants of the flat model, one for two- and one for 16-bar sequences, were trained and evaluated where the hierarchical model was only trained on 16-bar sequences. The two-bar model and the 16-bar models were trained with the Adam optimization algorithm and a learning rate annealed from 10^{-3} to 10^{-5} with exponential decay. The loss function can be written as

$$L(x,\hat{x}) = CE(x, p_\theta(X|z)) + \beta \max\left[D_{KL}\left(\mathcal{N}\left(\mu(x), \text{diag}\left(\sigma(x)^2\right)\right) \;||\, \mathcal{N}(0,I)\right),\; \lambda\right] \tag{4.2}$$

with (I. Goodfellow *et al.*, 2016a):

$$CE(p(X), q(X)) = -\sum_{x\in\mathcal{X}} p(x)\log(q(x)), \tag{4.3}$$

being the cross-entropy and (MacKay, 2003):

$$D_{KL}(p(X)||q(X)) = \sum_{x\in\mathcal{X}} p(x)\log\left(\frac{p(x)}{q(x)}\right). \tag{4.4}$$

being the KL divergence and z being sampled from the encoder's output distribution. The cross-entropy against the ground-truth excerpts is used as reconstruction loss. The second summand is the KL divergence with the output distribution of the encoder as the first, and the prior distribution as a second argument. The KL divergence is limited by a threshold λ, as suggested by Kingma and Welling (2019) and the parameter β is used to adjust the weight of the KL divergence term as the training proceeds, which is a technique proposed by Bowman *et al.* (2016).

The batch size was 512. The two-bar models were run for 50k steps and the 16-bar models for 100k steps. For two-bar sequences sequences, the model was trained with $\lambda = 33.3$ and β annealed from 0.0 to 0.2 with an exponential rate of 0.999 99. For 16-bar sequences, the model was trained with $\lambda = 177.4$, and $\beta = 0.2$.[4] For short-sequence models scheduled sampling with an inverse sigmoid rate of 2000 (S. Bengio *et al.*, 2015) was used and for long-sequence models, teacher forcing (Williams & Zipser, 1989). Teacher forcing is a specific technique to train a neural network. It means that the input token for the generation of $\hat{x}_t$ is not $\hat{x}_{t-1}$, as described for the generation process, but x_{t-1}. During application, the model then do not use teacher sequences as input, but its own output tokens. An advantage of this technique is that the RNN is forced to stay close to the sequence to be reconstructed (Lamb *et al.*, 2016). A drawback is the discrepancy between the type of input during training and during application. Scheduled sampling tries to overcome this issue by training the model using teacher forcing in the early stages of training and substituting teacher tokens with generated ones later on following a given probability. The inverse sigmoid rate determines how this probability changes over multiple training steps.

For reconstruction, Roberts *et al.* encoded and decoded a ground-truth excerpt x, yielding a reconstructed excerpt $\hat{x}$. The term *accuracy* refers to the percentage of values that are correct, in this case, the percentage of tokens in $\hat{x}$ that have exactly the same value at the corresponding time step in x. They found that the flat model lead to a high accuracy and did not suffer from posterior collapse for short sequences, but neither was the case for 16-bar sequences. They assumed that the cause of this is the vanishing influence of the latent vector as the output sequence is generated.The hierarchical model performed better than the flat model with long sequences in respect to accuracy and posterior collapse. The authors also conducted experiments with the 16-bar models where they interpolated between two vectors and manipulated attributes of an excerpt by performing specific operations on its latent vector. It was shown that the flat model was able to produce latent-space interpolation well and that the hierarchical model performed even better.

[4] The values of these parameters were retrieved from Roberts *et al.* (2018) and Magenta developers (2017a, ll. 78–104, 364–386).

Chapter 5: Implementation

The flat model of MusicVAE was re-implemented by using the original paper by Roberts *et al.* (2018) and their implementation (Magenta developers, 2017a, ll. 78–104) as a reference. However, a different dataset was used and the training was also performed differently. Details will be explained in the following sections.

Section 5.1: Data

As a dataset, the clean MIDI subset of the LMD created by Raffel (2016) was used. To retrieve the set of excerpts from the set of MIDI track files, a pre-processing pipeline similar to that of MusicVAE was used, as shown in Algorithm 1. It was assumed that

Algorithm 1: Pre-processing of the clean MIDI dataset

```
   Input: set of files in the clean MIDI dataset
 1 preprocessed_dataset := empty list;
 2 foreach song in LMD do
 3 |   read song using pretty_midi;   /* pretty_midi quantizes the
   |     song */
 4 |   continue with next song, if time signature of song is not 4/4;
 5 |   remove all drum tracks;
 6 |   remove empty tracks;
 7 |   foreach track in song do
 8 |   |   sequences := from track extracted sequences using sliding
   |   |     window of size of 2 bars and stride of 1 bar;
 9 |   |   remove polyphonic sequence;
10 |   |   remove sequences with rests longer than one bar;
11 |   |   append sequences to preprocessed_dataset;
12 |   end
13 end
14 deduplicate preprocessed_dataset;
15 return preprocessed_dataset;
```

every song started at a downbeat. However, this is not true in every case. Extending the pre-processing algorithm is left as an avenue for future experiments. To read and write MIDI files, as well as to do some processing tasks, multiple libraries were used,

among others `pretty_midi` (Raffel & Ellis, 2014) and `pypianoroll` (Dong *et al.*, 2018). The quantization policy used to quantify the melodies can be found in their documentations. Following that pre-processing procedure, 77 124 excerpts were retrieved for the dataset. Afterwards, it was split following the ratio 80 : 10 : 10 into training, validation and test sets, respectively. For data augmentation, each note sequence was transposed (meaning "shifted" in this context) by a random number of semitones sampled from a uniform distribution

$$\mathcal{U}(\max(-6, \texttt{min\_semitone\_shift}), \min(6, \texttt{max\_semitone\_shift})), \tag{5.1}$$

which covers exactly one octave (12 semitones), if the boundaries are not considered. The variable `min_semitone_shift` is the distance of the lowest pitch in an excerpt to 0 and `max_semitone_shift` is the distance from the highest pitch in an excerpt to 127[1]. This augmentation was applied every time an excerpt was drawn from the dataset during training. Its goal was to enable the network learning an invariance of key transpositions. The data are transposed only within one octave because specific pitch ranges might correspond to specific instruments or roles in a musical piece and therefore to certain characteristics of a melody that only occur in a limited pitch range.

Each excerpt from the pre-processed set was a two-bar sequence of 16^{th} notes. Each note was encoded as a one-hot vector of 129 elements, so one excerpt was a sequence of 32 vectors. A value 1 at index $i_p \in [0, 127]$ represents a note with a pitch encoded as a MIDI note number with $C_4 = 60$. A value 1 at index 128 represented a rest. If the sequence contained a sub-sequence of vectors with the same values, this was interpreted as one long note. Also note that therefore, repeated 16^{th} notes were essentially merged in the input. No distinction was made between bass voices and other melodies. An illustration of the representation used can be seen in Figure 5.1.

Section 5.2: Training

The same loss function that was used by Roberts *et al.* (2018) was chosen for the re-implementation. The KL divergence was treated as a regularization term. During training, teacher forcing was applied. This required to shift the ground-truth excerpt x one time step in the positive time direction, since the input at time step t to retrieve

[1] Two bugs were found in the code. They were since fixed, but re-training was not performed due to time constrains. First, if there was an empty token in the sequence, the range of the uniform distribution was $(\max(\texttt{min\_semitone\_shift}, -6), 2)$. The proportion of affected sequences in the training set is 0.6385. Second, notes in range 125 to 127 might have been rotated to be in the range 0 to 2 during transposition. The proportion of affected sequences in the training set is 0.6503×10^{-6}.

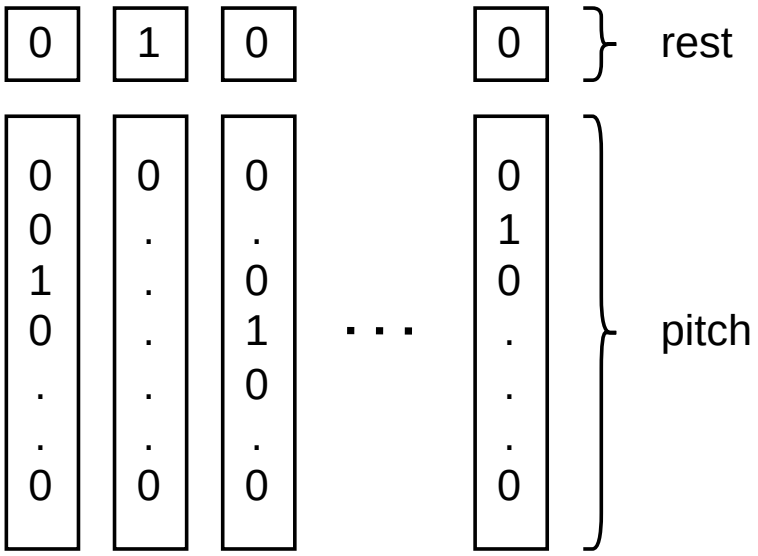


Figure 5.1: Representation of a note sequence.

$\hat{x}_t$ had to be x_{t-1} and not x_t. To make this possible, the representation was extended to a vector of dimension 130. The chosen start symbol, a one-hot vector with a value 1 at index 129, was then prepended before passing x as an input to the decoder. As in the work of Roberts *et al.* (2018), Adam (Kingma & Ba, 2015) was used as the optimization algorithm with an initial learning rate of 10^{-3}. The batch size used for training was 64. An L_2 weight decay (I. Goodfellow *et al.*, 2016b) was used with a factor of 10^{-6}.

Roberts *et al.* scheduled the learning rate using exponential decay. As a scheduling procedure in this work, the learning rate was halved when the validation reconstruction loss did not decrease for five epochs. If the validation reconstruction loss did not decrease after the learning rate was subsequently halved two times, the learning rate was reset to its initial value. The reason for this reset is explained in Section 6.1

All models were trained for at least 80 epochs. After the 50^{th} epoch passed, early stopping (Prechelt, 2012) was applied to stop training as soon as the validation reconstruction loss has not improved for 30 epochs. Early stopping was not applied before the 50^{th} epoch, because sudden increases of the reconstruction loss were observed, especially in the early epochs. If early stopping would have been applied in such early training stages, a stop of training would have been likely. However, a higher value than 30 would have led to a slower response time of the early stopping, so training would have been terminated much later than necessary. Based on the observations

of previous training experiments, the 50^{th} epoch seemed to be a good boundary to start the early stopping.

The learning rate was reset after two subsequent halvings of the learning rate without a decrease of the loss. The goal behind this technique was to find minima of the reconstruction loss with a low KL divergence faster. During the training of different models, sudden increases of the reconstruction loss followed again by a decrease were observed. Such events often occurred simultaneously with a sudden increase of the KL divergence. The following behavior was hypothesized: the model learned a local minimum where the reconstruction loss was low, but the latent space not well-formed. This was penalized by the KL divergence term. Following gradient updates led to weight configurations that made the reconstruction loss increase significantly. To enable the model to quickly find new regions in the loss space where both reconstruction loss and KL divergence are relatively low, the learning rate was increased after such an event. This decision, however is based on a hypothesis that is difficult to test, so that will be left for future work.

Chapter 6: Experiments and Discussion

Before generating sequences, a grid search over two hyperparameters was performed, based on which one model was chosen for generation and thus further evaluation. As a part of that, the quality of the latent space was evaluated. The question if the generated music has good quality was interpreted as "Is the generated data original?" and "Is the training data reconstructed well?" where the main focus was on the second question. These questions were evaluated using objective metrics and a subjective analysis. A more thorough evaluation, *e.g.* a larger-scale user study would be an avenue for future work.

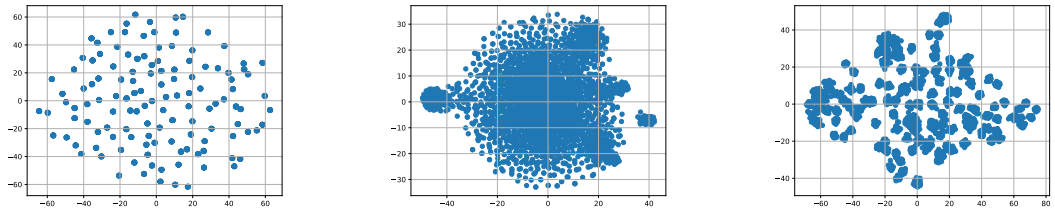
Section 6.1: Evaluation of the Training

Unlike in the training of MusicVAE, here, the parameters of the KL divergence term β and λ were set to constant values and not annealed over steps. A grid search was performed to find good values for them. The values that were tried are listed in Table 6.1. For the resulting curves of each model, such as learning rate or reconstruction loss, see Appendix A.

When training Model 3, the average KL divergence shown in Figure A.3e was below its threshold $\lambda = 333$ all the time. This led to the KL divergence not being

Table 6.1: Values of hyperparameters that have been chosen for the grid search.

Model number	β	λ
1	0.01	3.33
2	0.01	33.3
3	0.01	333
4	0.10	3.33
5	0.10	33.3
6	0.10	333
7	1.00	3.33
8	1.00	33.3
9	1.00	333



(a) Model 1: $\beta = 0.01$, $\lambda = 3.33$

(b) Model 7: $\beta = 1.00$, $\lambda = 3.33$

(c) Model 8: $\beta = 1.00$, $\lambda = 33.3$

Figure 6.1: Samples from the encoder's output distribution. The dimensionality was reduced from 512 to 50 using **p**rincipal **c**omponent **a**nalysis (PCA) (Gewers *et al.*, 2021) and afterwards, from 50 to 2 using ***t***-distributed **s**tochastic **n**eighbor **e**mbedding (*t*-SNE) (van der Maaten & Hinton, 2008). The algorithms were implemented using Scikit-learn (Pedregosa *et al.*, 2011) with standard parameters. The reason, why both procedures were applied subsequently was a suggestion in the Scikit-learn documentation.

penalized at all. The value $\lambda = 333$ was therefore considered as too high in general and Models 3, 6, and 9 were not examined to a larger extent.

The Models 1, 7, and 8 were selected for further evaluation. Model 1 had the lowest reconstruction loss (see Figure A.2), but a higher KL divergence than other models (see Figure A.3). Model 7 has the lowest KL divergence, but a higher reconstruction loss than other models. Model 8 had the third lowest reconstruction loss and the second lowest KL divergence, so it is the model with the lowest reconstruction loss where both reconstruction loss and KL divergence are relatively low.

In all of the trained models, the training loss was below the validation loss. This implies that they were overfit. Overfit models perform well on the excerpts they were trained with, but when it comes to generalization, they are worse than in training (I. Goodfellow *et al.*, 2016b). To avoid this effect, different methods could be applied, *e.g.* fine tuning of the weight decay parameter, which will be left for future work.

To evaluate the quality of the latent space, one could conduct interpolation experiments and evaluate the interpolated excerpts, as performed by Roberts *et al.* Another option is to directly measure or visualize features of the resulting latent space, which was done here. From the test dataset, 128 samples were drawn randomly and fed into the encoder, yielding a set of Gaussians. From each Gaussian, 20 samples were drawn. The result was a cloud of 2560 points in a 512-dimensional space. In Figure 6.1 these points were visualized for each of the Models 1, 7, and 8. However, these plots cannot be used for evaluation, because the high amount of information that probably was lost during the dimensionality reduction process.

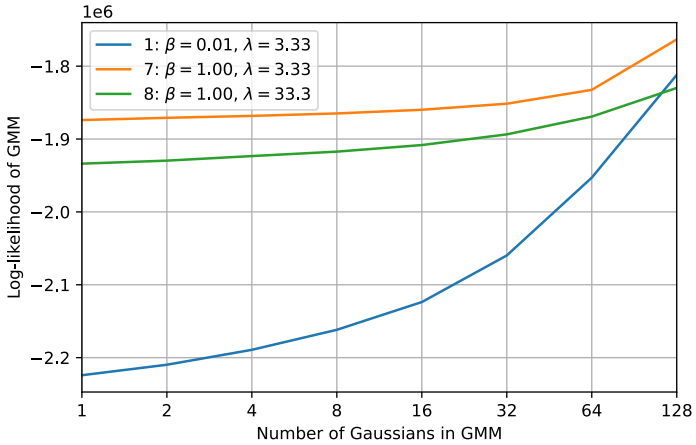


Figure 6.2: The log-likelihood of a GMM with a varying number of Gaussians fitted to the latent spaces of Models 1, 7, and 8.

To be able to make quantitative statements, multiple **g**aussian **m**ixture **m**odels (GMMs) (Reynolds & Jain, 2009) were fitted onto the same set of 512-dimensional points using Scikit-learn (Pedregosa *et al.*, 2011). Each of those GMMs consisted of a different number of Gaussians. The number of Gaussians was a power of two, ranging from $2^0 = 1$ to $2^7 = 128$. Each Gaussian was assumed to have a diagonal covariance matrix. For each GMM, the log-likelihood of each point to be sampled from that GMM was calculated and the log-likelihoods of all points were summed up to retrieve the log-likelihood of the GMM. In that way, a function between the number of Gaussians in the GMM and its log-likelihood was retrieved. This procedure was repeated for each of the Models 1, 7, and 8. The log-likelihood curves can be seen in Figure 6.2. First, it can be seen that in all cases, the likelihood is higher the more Gaussians there are in the GMM. The reason for that is that the points are sampled from 128 different Gaussians. Second, a GMM with 128 Gaussians can approximate the distribution of these data better than a single Gaussian distribution, so the log-likelihood raises with the number of Gaussians. Third, the curve of Model 1 is much lower for 1 Gaussian in the GMM than the remaining curves. Only with 64 to 128 Gaussians the log-likelihood for Model 1 reaches the level of the other log-likelihood curves. A reason for this might be that the latent space of Model 1 is less similar to a Gaussian distribution than the latent space of the Models 7 and 8. In other words, Model 1 has a latent space containing more holes than the latent spaces of Models 7 and 8. The log-likelihood curve for Model 8 is slightly lower than that of Model 7. The latent space of Model 1 was not formed well, compared to Models 7 and 8. Therefore, Model 1 was ruled out for further evaluation, leaving only Models 7 and 8.

To choose one of the remaining models for further evaluation, several sequences were generated. A subjective analysis revealed that Model 8 produced a higher proportion of sequences that sound pleasing. For this reason, Model 8 was examined in more detail, where only that weight configuration has been looked at that had the lowest validation reconstruction loss.

Section 6.2: Quality of the Generated Excerpts

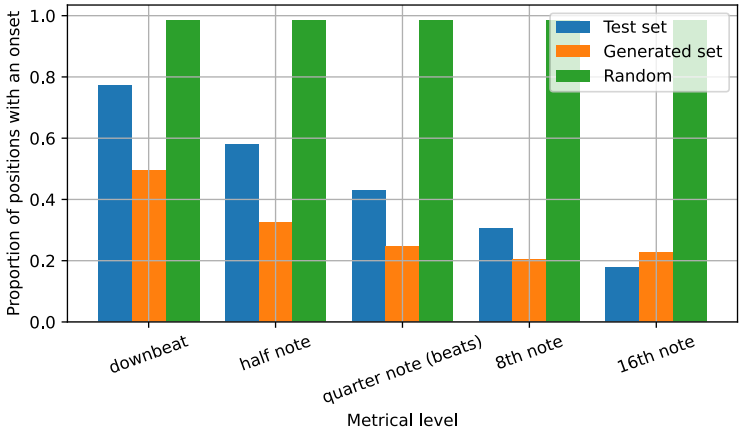
To answer the question if the training data is reconstructed well, several evaluation measures were be selected and calculated for the training set, a set of generated excerpts (called *generated set*), and a set of random sequences (called *random set*). The excerpts retrieved from the model were produced by sampling latent vectors from a 512-dimensional multivariate normal distribution with $\mu = 0$ and $\sigma = 1.2802$ and passing those to the decoder. These values were retrieved by fitting a single spherical Gaussian onto the point cloud mentioned in Section 6.1, where the GMM from Scikit-learn (Pedregosa *et al.*, 2011) was used. Each sequence in the random set was generated by sampling each one-hot index uniformly and converting them to a one-hot vector. The goal was to see if the model is able to reconstruct rhythmic and melodic features.

For the evaluation of the originality of the generated music a rather simple metric was used: for each excerpt it was checked whether it was an exact copy of an excerpt from the training dataset. There were 106 duplicate excerpts in the generated dataset itself, which is a proportion of 0.0135. After de-duplication of the generated dataset, it contained 35 excerpts that were also in the training dataset, which is a proportion of 0.0045. The training dataset did not contain any duplicates. More thorough evaluations, as described in Section 3.5, are left for future work.

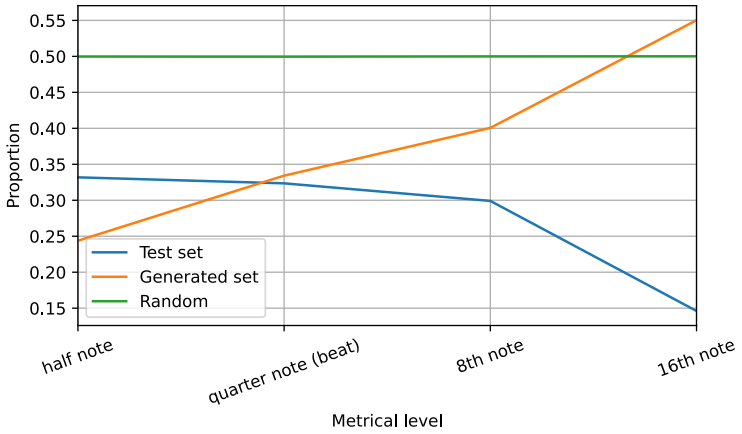
Section 6.2.1: Rhythmic Features

Three rhythmic features were selected for comparison: the proportion of metrical positions with an onset and the distribution of note lengths, both calculated over the whole set. Furthermore, the distribution of the average note length per set will be investigated.

Figure 6.3a shows the proportion of metrical positions of each metrical level that contain an onset. It can be seen that the random set had many onsets at almost every metrical position, regardless of the level. This indicates that the random note sequences contained mostly 16^{th} notes. Looking at the test set and the generated set, it can be seen that on downbeats the proportion of onsets is higher than at other metrical



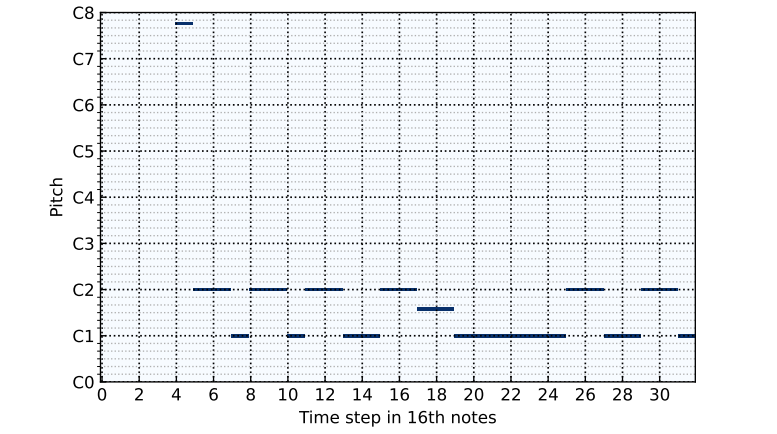
(a) Proportion of metrical positions with an onset for different metrical levels.



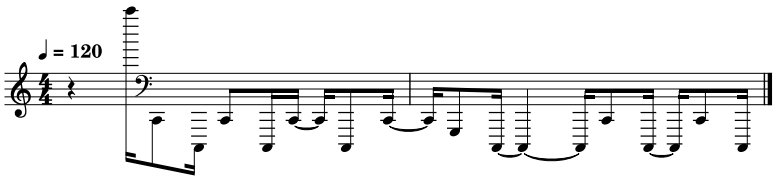
(b) Proportion of onsets that are on a specific metrical level, but not on the previous metrical level.

Figure 6.3: Onset proportions.

levels. The generated set shows a similar trend as the test set. However, the proportions are lower. In contrast to the test set, in the generated set the proportions of 16^{th} notes with an onset is slightly higher than the proportion of eighth notes with an onset. This suggests that there were more onsets on uneven 16^{th} notes than on even ones. This property is illustrated by the generated excerpt shown in Figure 6.4. Figure 6.3b, shows the proportions of onsets that are on a specific metrical level, but not on its previous one. From that figure, it can even be seen that this proportion increases over the metrical levels, where the curve of the test set decreases. The property described for 16^{th} notes thus applies to the other metrical levels too.

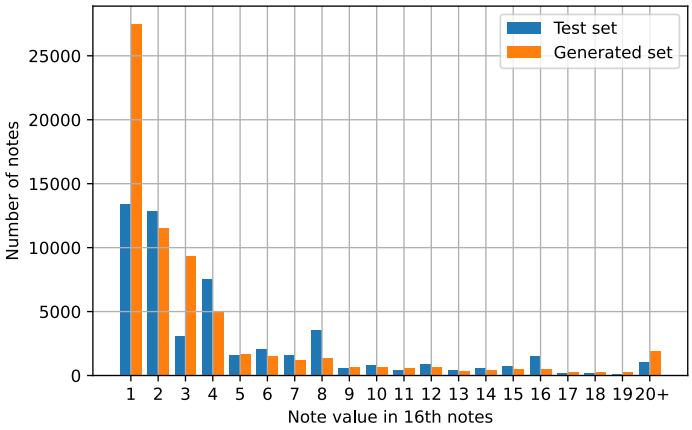


(a)

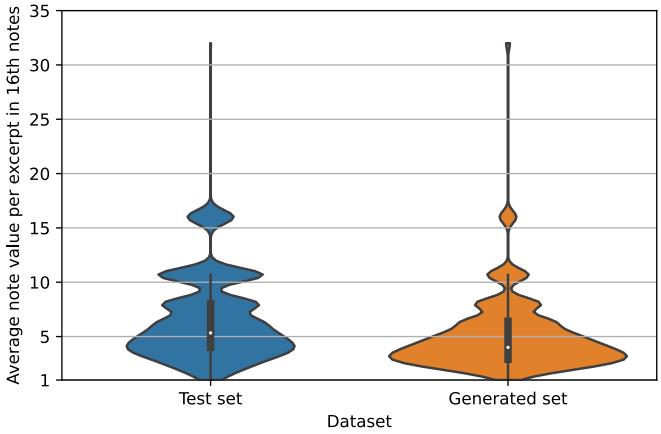


(b)

Figure 6.4: Generated sequence where most of the onsets are on uneven 16^{th} notes. Figure 6.4a and Figure 6.4b both show the same sequence, first as piano roll and second as score.



(a) Bar plot of how many notes in a set have which note length.



(b) Distribution of the average note length of a sequence per set.

Figure 6.5: Note length.

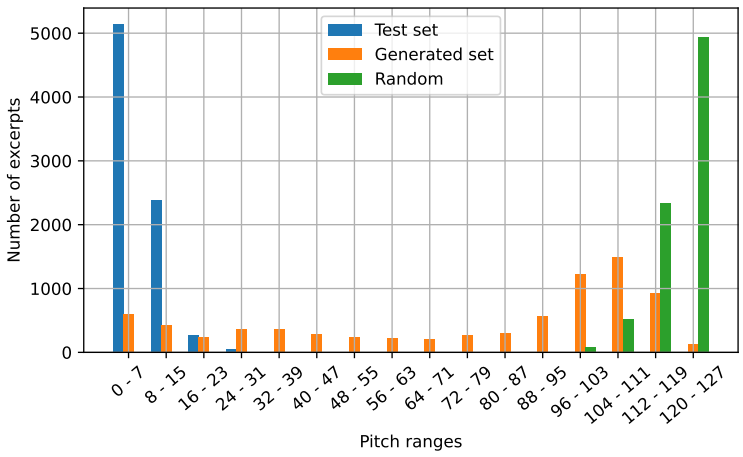


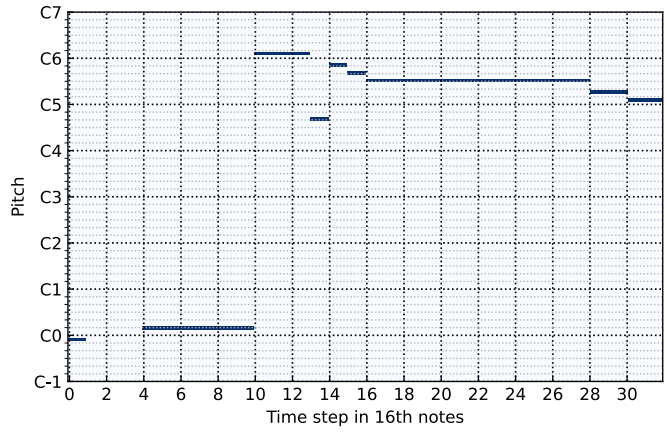
Figure 6.6: Plot of how many samples in each set have what pitch range.

Figure 6.5a shows the distribution of note length over all notes in a set. It can be seen that the number of notes decreases smoothly the higher of the duration of the notes becomes. The test set exhibits peaks at 16^{th}, eighth, quarter, half, and whole notes (the corresponding x-axis labels are 1, 2, 4, 8, and 16). The generated set, in contrast, shows only the smooth decrease over increase of note duration. The peaks were not copied, nor was the low number of notes with a duration of three 16^{th} notes present. A possible reason is that the training dataset was not big enough for the model to learn these specific features, especially because the number of longer notes is very small compared to the number of overall notes in the set.

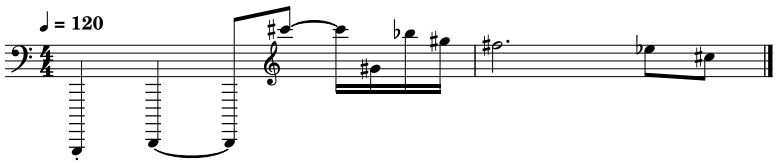
Figure 6.5b shows the distribution of the average note length of a sequence. In the graph of the test set there are peaks around average note lengths of approximately four, eight, eleven, and 16 16^{th} notes. The maxima in the distribution of the test set correspond to sequences with few longer notes taking most of the space and maybe some shorter ones before or afterwards. The distribution of the generated set has a higher peak at approximately four 16^{th} notes than the test set. This suggests that the generated set contained a higher amount of sequences with mostly short notes.

Section 6.2.2: Melodic Features

To compare melodic properties, two features were be taken into account: the pitch range and a measure for diatonicity. For each excerpt in a set, the pitch range was calculated, leading to a list of scalar values, each representing the pitch range of an excerpt. The distribution of these pitch ranges are shown in Figure 6.6 for each set. It can be seen that almost all excerpts in the Test set had a pitch range in the interval

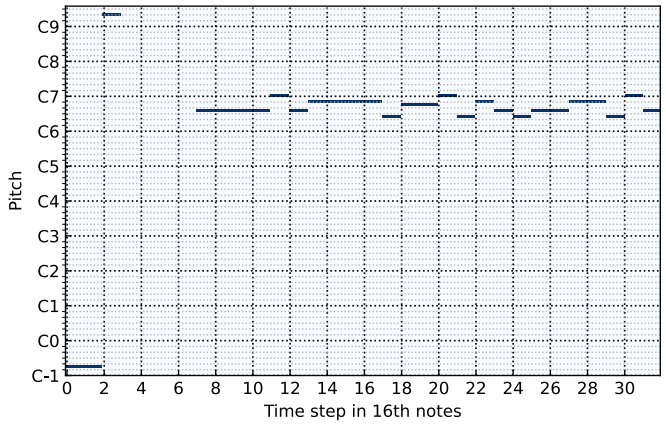


(a)

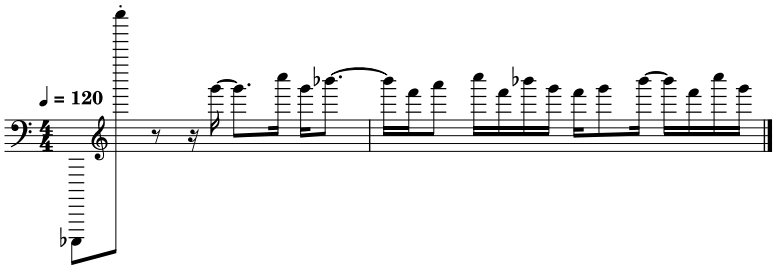


(b)

Figure 6.7: Generated sequence that starts with notes that lie significantly outside a two-octave pitch range and then settles down to a more conservative melody. Figure 6.7a and Figure 6.7b both show the same sequence, first as piano roll and second as score.

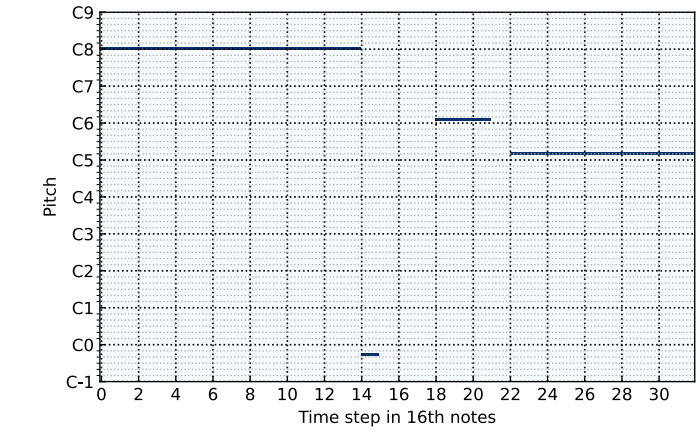


(a)

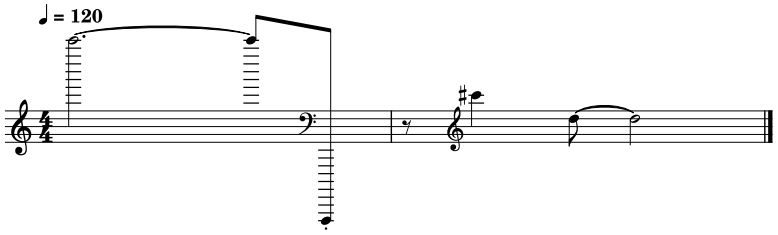


(b)

Figure 6.8: Generated sequence that starts with notes that lie significantly outside a two-octave pitch range and then settles down to a more conservative melody. Figure 6.8a and Figure 6.8b both show the same sequence, first as piano roll and second as score.



(a)



(b)

Figure 6.9: Generated sequence that contains a note in the middle of it that lies significantly outside a two-octave pitch range. Figure 6.9a and Figure 6.9b both show the same sequence, first as piano roll and second as score.

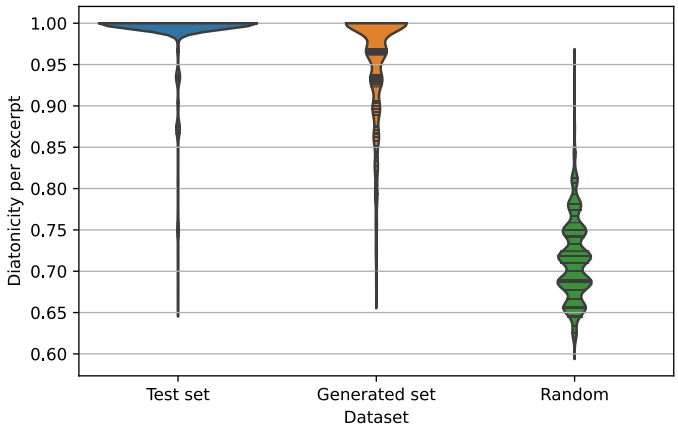
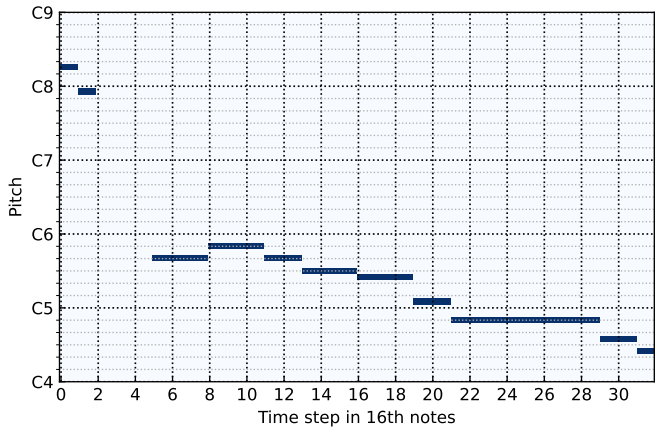
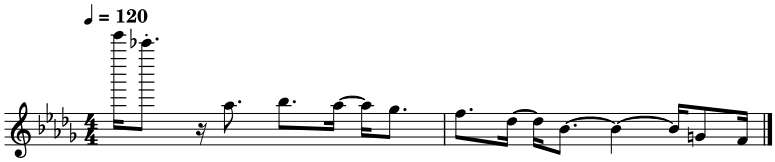


Figure 6.10: Plot of which proportion of excerpts had which diatonicity value. The diatonicity value was computed by calculating the vector products of different binary pitch class vector templates (one for each major key) and the pitch class vector of the excerpt and taking the maximum of those vector products.



(a)



(b)

Figure 6.11: Generated sequence that is estimated to be in key $F\sharp$ with a score of 0.9310 containing two notes not in that key: the 29^{th} and the 30^{th}. Figure 6.11a and Figure 6.11b both show the same sequence, first as piano roll and second as score.

$[0, 23]$, so the pitches of most songs were placed within one or two octaves. It can be seen that the majority of generated excerpts had high pitch range, similar to the random set. Reasons for such high pitch ranges in the generated set are sequences that start with notes that definitely exceed the pitch range of two octaves, but eventually settle to a more conservative range. Such sequences are shown in Figure 6.7 and Figure 6.8. There were also excerpts that contained notes in the middle of the sequence, which lie significantly outside a two-octave range, such as the sequence shown in Figure 6.9.

To retrieve a measure for diatonicity, a pitch-class vector was calculated for each excerpt. The pitch-class vector contained for each pitch class the number of non-rest tokens that correspond to that specific pitch class, normalized by the number of non-rest tokens in a sequence. Next, a key-estimation vector was to be calculated. To achieve that, a binary pitch class template for major scales was defined, containing twelve elements, one for each pitch class. Let the first index of the template correspond to a root note of a scale. The template was created by setting the value to one at those indices that correspond to a semitone that is in the major scale of the root note. The retrieved template then applied to an arbitrary root note:

$$\text{binary major pitch class template} = (1,0,1,0,1,1,0,1,0,1,0,1). \tag{6.1}$$

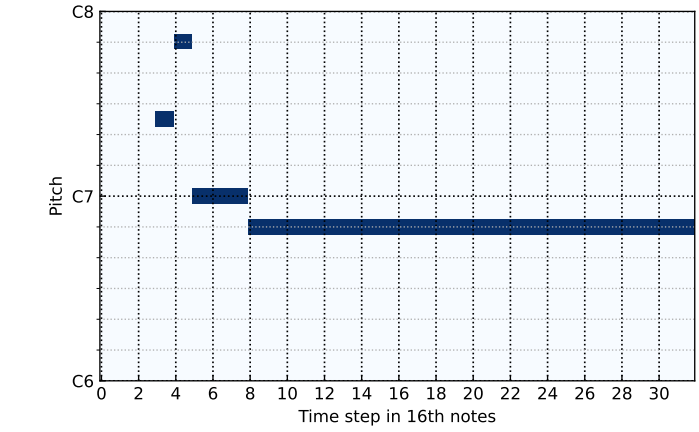
The key estimation vector was retrieved by calculating the dot product of the considered piece's pitch class vector and every possible rotation of the pitch class template. The resulting key estimation vector was a list of scores, determining which major key fitted how well to the piece. Since every major key has a relative natural minor key (the relative minor key of C major is A minor), which contains the same pitches as its relative major key, this procedure also included all possible natural minor keys implicitly. For each piece in a set, the maximum value of that key estimation vector was taken for evaluation and called *diatonicity value*. These values can be seen in Figure 6.10.. From the plot it can be seen that the random set had rather low diatonicity values, where the generated set and the test set contained most excerpts with a diatonicity value of 1. Still, the distribution of the generated set looks not as steep as that of the test set, there were more excerpts with lower diatonicity, than in the test set[1]. The reason for that might be that the generated set contained excerpts where most notes sound pleasant and diatonic, but some notes did not fit to the rest. One typical example is shown in

[1] The reason why the distributions are rippled is that without normalizing the pitch class vector by the number of non-rest notes, which can be different for multiple excerpts, the diatonicity score takes distinct values. With the normalization, however, the number of possible distinct values is much higher, hence violin plots were used for visualization.

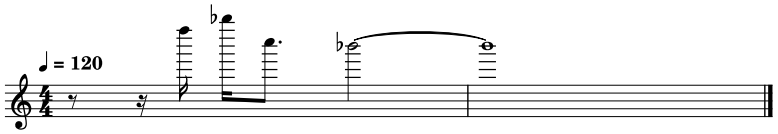
Figure 6.11. It had a maximum key score of 0.9310 for the key $F\sharp$ and it contained two tokens that are out of key: the 29^{th} and the 30^{th}.

Section 6.2.3: Qualitative Evaluation

For qualitative evaluation, several excerpts were listened to manually. Many excerpts showed sudden changes in the scale or big jumps in the pitch that exceed pleasant sounding pitch ranges (see Section 6.2.2). Rhythmic differences were not noticed that much from a subjective point of view. The reason might be that the excerpts were only monophonic without having any group of rhythmic voices playing along. A sequence shifted by some steps thus sounding more syncopated would not have been noticed. Apart from these non-musical properties that distinguished the generated set from the test set, some good sounding excerpts were generated too. Ignoring the few odd notes at the beginning or the end, the sequences shown in Figure 6.11 and Figure 6.7 sounded pleasant and were coherent in their theme throughout the two bars. Furthermore, cheerful chimes (a few short notes followed by a final long note, all with a relatively high pitch) were generated, such as Figure 6.12 and Figure 6.13 that could easily be imagined becoming a jingle for a spot for *e.g.* a private television channel. Listening to approximately fifty excerpts, around five of those chimes were found. Rhythmically interesting are the sequences shown in Figure 6.14 and Figure 6.15. Especially the second one is a pleasant sounding bass voice that one could imagine appearing in a progressive rock song. Such rhythmical interesting excerpts were found around three times in approximately fifty excerpts.

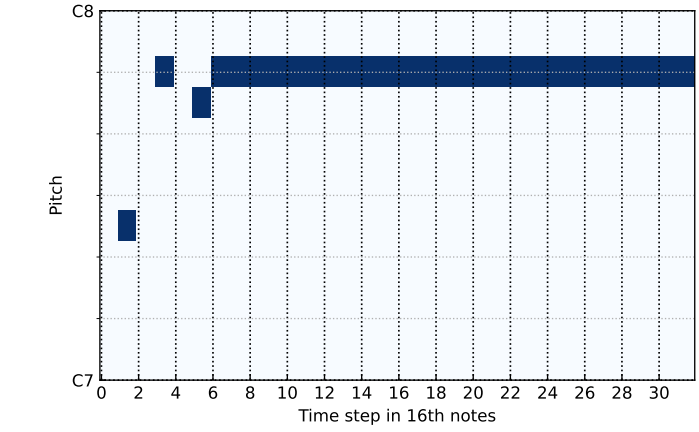


(a)

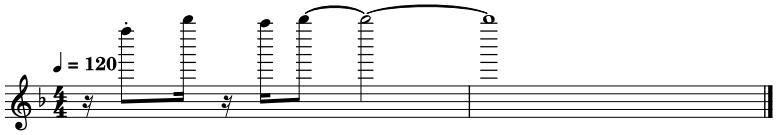


(b)

Figure 6.12: Generated sequence that is an example for a chime. Figure 6.12a and Figure 6.12b both show the same sequence, first as piano roll and second as score.

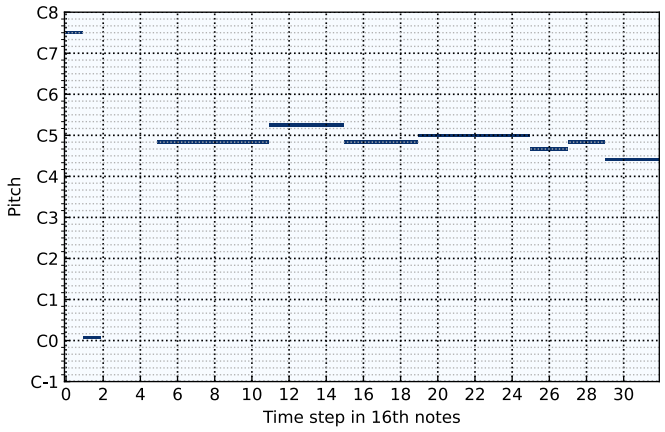


(a)

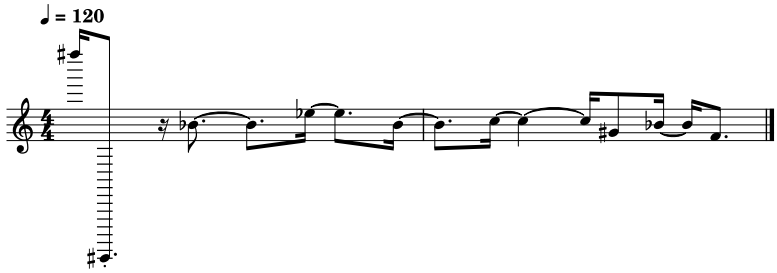


(b)

Figure 6.13: Generated sequence that is an example for a chime. Figure 6.13a and Figure 6.13b both show the same sequence, first as piano roll and second as score.

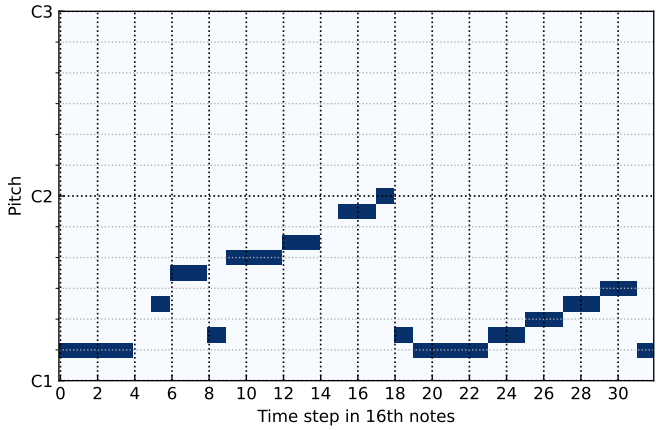


(a)

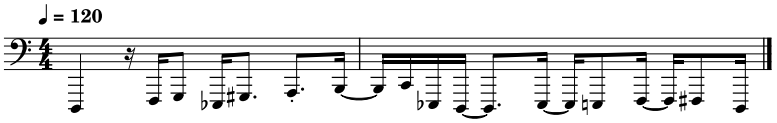


(b)

Figure 6.14: Generated sequence that shows a rhythmically interesting melody. Figure 6.14a and Figure 6.14b both show the same sequence, first as piano roll and second as score.



(a)



(b)

Figure 6.15: Generated sequence that shows a rhythmically interesting bass voice. Figure 6.15a and Figure 6.15b both show the same sequence, first as piano roll and second as score.

Chapter 7: Conclusion and Future Work

In this thesis, the task of unconditioned generation of symbolic monophonic music was considered. The state of the art to this task was investigated, including commonly used music representations and evaluation protocols. A selected variant of the music generation system MusicVAE (Roberts *et al.*, 2018) was re-implemented to generate a set of two-bar monophonic note sequences. The quality of these excerpts was then discussed using objective metrics.

The re-implemented model was able to reduce its reconstruction loss during training with several hyper-parameter configurations. However, in every case the model overfit. To choose one of the trained models for further evaluation, the latent spaces of different models were compared and a subjective analysis on the quality of the models' output was performed. That model that showed a good trade-off between qualities of latent space and reconstruction was considered for the following analysis.

With that model, a set of generated sequences was created and compared to the test set for evaluating the reconstruction quality and to the training set for evaluating the originality. It was found that the model did not just learn to copy excerpts from the test set to a noticeable amount, but generated new ones. The generated excerpts were more similar to those in the test set than randomly generated sequences. However, there were some differences: the model struggled to reproduce the rhythmic structure of the training data. In particular, it had to many notes with a lower duration and characteristic peaks in the note duration distribution were not recognized. There were more onsets on uneven 16^{th} notes in the generated set than in the test set. Considering melodic features, the generated sequences contained erroneous sub-sequences. In particular, there were single non-diatonic notes relative to the majority of notes in the sequence and sudden note jumps that exceeded a pleasant pitch range. However, except those odd sub-sequences, most of the notes in a sequence were diatonic and the model was capable of producing interesting rhythms, chimes and other melodies.

For further research, the dataset creation procedure can be improved. First, a larger corpus of MIDI files (such as the full LMD) could be collected to retrieve a larger set of excerpts for training and it can be evaluated if that solves the problem of the model not copying characteristic rhythmical features. The extraction of the

sequences can be extended to detect the downbeats of songs that do not have their downbeat at the first time step. After excerpt creation, a thorough analysis of the retrieved dataset can be beneficial to make sure that different types of excerpts have a similar proportion. Such a type could be a kind of voice or a class of melodies, *e.g.* bass voice, lead melody, chime, *etc.* Alternative representations of symbolic music might be considered, *e.g.* where a note is encoded with an onset token and some information about the duration. Furthermore, the representation could include an encoding where the distance of two notes of other features than only the pitch, such as the proximity on the circle of fifths, as in the work of Mozer (1994). To reduce possible overfitting, regularization techniques, such as weight decay and dropout might be applied and the corresponding hyper-parameters tuned. The annealing of the hyper-parameter β of the KL divergence term, as performed by Roberts *et al.* (2018), could be applied, scheduled sampling could be tried instead of teacher forcing, and a more thorough hyper-parameter search could be conducted. Sudden jumps in the loss curve were observed during training, but they were not investigated thoroughly. Future work could include an analysis of the reasons for such events. That knowledge could then be used to improve the training procedure.

In this thesis, only the baseline from the work of Roberts *et al.* (2018) was re-implemented. This could also be done for their hierarchical model to be able to generate longer sequences incorporating a coherent structure over a higher amount of time steps. As a next step, the generation scheme of the model could be extended to generate polyphonic music directly, not only multiple streams of monophonic voices. The performance of model variants with different sizes of the latent space can also be investigated, as well as ways to condition the latent space. One possibility for that would be to learn a mapping from text labels to excerpts using an additional loss function to urge the learned latent space to be close to the text label's embedding. Other evaluation procedures may be performed, such as larger-scale listening tests, experiments involving more thorough objective metrics, as listed by Yang and Lerch (2020) and a more sophisticated originality report, as proposed by Yin *et al.* (2023). Further possibilities to evaluate the quality of the latent space could also be an avenue for further research.

Bibliography

Agostinelli, A., Denk, T. I., Borsos, Z., Engel, J., Verzetti, M., Caillon, A., Huang, Q., Jansen, A., Roberts, A., Tagliasacchi, M., Sharifi, M., Zeghidour, N., & Frank, C. (2023). MusicLM: Generating music from text. *arXiv preprint arXiv:2301.11325*.

Bengio, S., Vinyals, O., Jaitly, N., & Shazeer, N. (2015). Scheduled sampling for sequence prediction with recurrent neural networks. *Proceedings of the 28th International Conference on Neural Information Processing Systems*, *1*, 1171–1179.

Bengio, Y., Ducharme, R., Vincent, P., & Janvin, C. (2003). A neural probabilistic language model. *The Journal of Machine Learning Research*, *3*, 1137–1155.

Bond-Taylor, S., Lerch, A., Long, Y., & Willcocks, C. G. (2022). Deep generative modelling: A comparative review of VAEs, GANs, normalizing flows, energy-based and autoregressive models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *44*(11), 7327–7347.

Bottou, L. (1999). On-line learning and stochastic approximations. In *On-line learning in neural networks* (pp. 9–42). Cambridge University Press.

Bowman, S. R., Vilnis, L., Vinyals, O., Dai, A., Jozefowicz, R., & Bengio, S. (2016). Generating sentences from a continuous space. *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, 10–21.

Briot, J.-P. (2021). From artificial neural networks to deep learning for music generation: History, concepts and trends. *Neural Computing and Applications*, *33*(1), 39–65.

Brunner, G., Konrad, A., Wang, Y., & Wattenhofer, R. (2018). MIDI-VAE: Modeling dynamics and instrumentation of music with applications to style transfer. *Proceedings of the 19th International Society for Music Information Retrieval Conference*, 747–754.

Cambridge University Press & Assessment. (2023). “music, noun”. In *Cambridge dictionary: English dictionary*. Retrieved June 25, 2023, from https://dictionary.cambridge.org/dictionary/english/

Cinelli, L. P., Araújo Marins, M., Barros da Silva, E. A., & Lima Netto, S. (2021a). Variational autoencoder. In *Variational methods for machine learning with applications to deep networks* (pp. 111–149). Springer International Publishing.

Cinelli, L. P., Araújo Marins, M., Barros da Silva, E. A., & Lima Netto, S. (2021b). *Variational methods for machine learning with applications to deep networks*. Springer International Publishing.

Collins, T., & Laney, R. (2017). Computer-generated stylistic compositions with long-term repetitive and phrasal structure. *Journal of Creative Music Systems*, *1*(2).

Creative Commons. (2021). *Should CC-licensed content be used to train AI? it depends*. Retrieved June 30, 2023, from https://creativecommons.org/2021/03/04/should-cc-licensed-content-be-used-to-train-ai-it-depends/

Dong, H., Hsiao, W., & Yang, Y. (2018). MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment. *Proceedings of the AAAI Conference on Artificial Intelligence*, *32*(1), 34–41.

Dong, H.-W., Hsiao, W.-Y., & Yang, Y.-H. (2018). Pypianoroll: Open source Python package for handling multitrack pianorolls. *Late-Breaking Demos of the 19th International Society for Music Information Retrieval Conference*.

Eck, D., & Schmidhuber, J. (2002). Learning the long-term structure of the blues. *International Conference on Artificial Neural Networks*, 284–289.

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biol Cybernetics*, *36*, 193–202.

Gagniuc, P. A. (2017). *Markov chains*. John Wiley & Sons, Ltd.

Gers, F., Schmidhuber, J., & Cummins, F. (1999). Learning to forget: Continual prediction with LSTM. *1999 Ninth International Conference on Artificial Neural Networks*, *2*, 850–855.

Gewers, F. L., Ferreira, G. R., Arruda, H. F. D., Silva, F. N., Comin, C. H., Amancio, D. R., & Costa, L. D. F. (2021). Principal component analysis: A natural approach to data exploration. *ACM Comput. Surv.*, *54*(4).

Goodfellow, I., Bengio, Y., & Courville, A. (2016a). *Deep learning*. MIT Press.

Goodfellow, I., Bengio, Y., & Courville, A. (2016b). Machine learning basics. In *Deep learning* (pp. 96–161). MIT Press.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial networks. *arXiv preprint arXiv:1406.2661v1*.

Gotham, M., Gullings, K., Hamm, C., Hughes, B., Jarvis, B., Lavengood, M., & Peterson, J. (2021). *Open music theory* (2nd ed.). VIVA Open Publishing. Retrieved June 25, 2023, from https://viva.pressbooks.pub/openmusictheory/

Hadjeres, G., Pachet, F., & Nielsen, F. (2017). DeepBach: A steerable model for Bach chorales generation. *Proceedings of the 34th International Conference on Machine Learning*, *70*, 1362–1371.

Hernandez-Olivan, C., Hernandez-Olivan, J., & Beltran, J. R. (2022). A survey on artificial intelligence for music generation: Agents, domains and perspectives. *arXiv preprint arXiv:2210.13944v2*.

Hiller, L. A., & Isaacson, L. M. (1959). *Experimental music; composition with an electronic computer* (1st). McGraw-Hill.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Huang, C.-Z. A., Cooijmans, T., Roberts, A., Courville, A. C., & Eck, D. (2017). Counterpoint by convolution. *Proceedings of the 18th International Society for Music Information Retrieval Conference*, 211–218.

Huang, C. A., Vaswani, A., Uszkoreit, J., Shazeer, N., Hawthorne, C., Dai, A. M., Hoffman, M. D., & Eck, D. (2018). Music transformer. *arXiv preprint arXiv:1809.04281v3*.

Jiang, J., Xia, G. G., Carlton, D. B., Anderson, C. N., & Miyakawa, R. H. (2020). Transformer VAE: A hierarchical model for structure-aware and interpretable music representation learning. *2020 IEEE International Conference on Acoustics, Speech and Signal Processing*, 516–520.

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. *Proceedings of the International Conference on Learning Representations*.

Kingma, D. P., & Welling, M. (2014). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kingma, D. P., & Welling, M. (2019). An introduction to variational autoencoders. *Foundations and Trends in Machine Learning*, *12*(4), 307–392.

Kostka, S., Payne, D., & Almen, B. (2018). *Tonal harmony*. McGraw Hill.

Lamb, A. M., ALIAS PARTH GOYAL, A. G., Zhang, Y., Zhang, S., Courville, A. C., & Bengio, Y. (2016). Professor forcing: A new algorithm for training recurrent networks. *Advances in Neural Information Processing Systems*, *29*.

Lehrmann, P. D. (1993). *MIDI for the professional*. Amsco Publications.

MacKay, D. J. C. (2003). Probability, entropy, and inference. In *Information theory, inference, and learning algorithms* (pp. 22–47). Cambridge University Press.

Magenta developers. (2017a). *File "config.py" in MusicVAE source code* (Version (commit) 78661d8) [Source code]. Retrieved May 30, 2023, from https://github.com/magenta/magenta/blob/main/magenta/models/music_vae/configs.py

Magenta developers. (2017b). *MusicVAE: A hierarchical latent vector model for learning long-term structure in music. jupyter notebook*. Retrieved May 31, 2023, from https://colab.research.google.com/github/magenta/magenta-demos/blob/master/colab-notebooks/MusicVAE.ipynb

Magenta developers. (2018). *MusicVAE: Creating a palette for musical scores with machine learning*. Retrieved June 27, 2023, from https://magenta.tensorflow.org/music-vae

Morris, R. O. (1975). *Introduction to counterpoint*. Oxford University Press.

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

Olah, C. (2015). *Understanding lstm networks*. Retrieved June 30, 2023, from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Prechelt, L. (2012). Early stopping – But when? In *Neural networks: Tricks of the trade: Second edition* (pp. 53–67). Springer Berlin Heidelberg.

Raffel, C., & Ellis, D. P. W. (2014). Intuitive analysis, creation and manipulation of MIDI data with pretty_midi. *15th International Conference on Music Information Retrieval Late Breaking and Demo Papers*.

Raffel, C. (2016). *Learning-based methods for comparing sequences, with applications to audio-to-MIDI alignment and matching* (Doctoral dissertation). Columbia University. New York, New York, USA.

Reynolds, D., & Jain, A. (2009). Gaussian mixture models. In *Encyclopedia of biometrics* (pp. 659–663). Springer US.

Roberts, A., Engel, J., Raffel, C., Hawthorne, C., & Eck, D. (2018). A hierarchical latent vector model for learning long-term structure in music. In J. Dy & A. Krause (Eds.), *Proceedings of the 35th international conference on machine learning* (pp. 4364–4373).

Sak, H., Senior, A., & Beaufays, F. (2014). Long short-term memory recurrent neural network architectures for large scale acoustic modeling. *Proc. Interspeech 2014*, 338–342.

Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, *45*(11), 2673–2681.

Todd, P. M. (1989). A connectionist approach to algorithmic composition. *Computer Music Journal*, *13*(4), 27–43.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., & Kavukcuoglu, K. (2016). WaveNet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, *9*(86), 2579–2605.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in neural information processing systems* (pp. 5998–6008). Curran Associates, Inc.

Williams, R. J., & Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, *1*(2), 270–280.

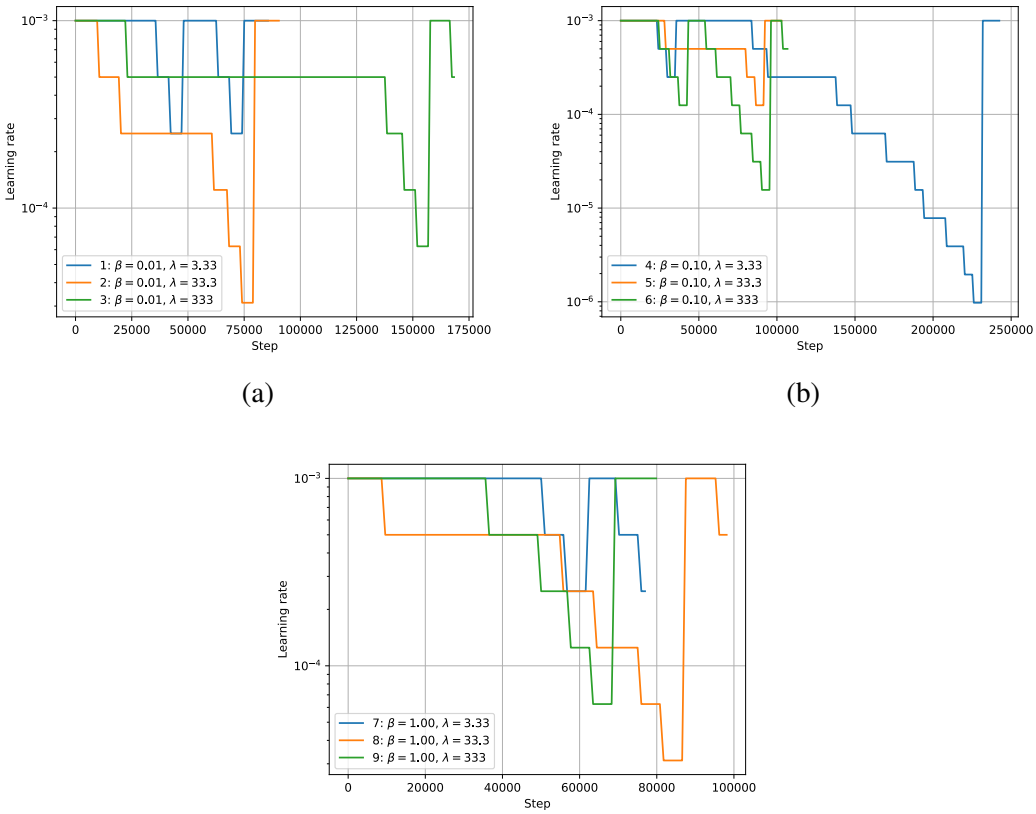
Yang, L., Chou, S., & Yang, Y. (2017). MidiNet: A convolutional generative adversarial network for symbolic-domain music generation. *Proceedings of the 18th International Society for Music Information Retrieval Conference*, 324–331.

Yang, L.-C., & Lerch, A. (2020). On the evaluation of generative models in music. *Neural Computing and Applications*, *32*(9), 4773–4784.

Yin, Z., Reuben, F., Stepney, S., & Collins, T. (2022). Measuring when a music generation algorithm copies too much: The originality report, cardinality score, and symbolic fingerprinting by geometric hashing. *SN Computer Science*, *3*(5).

Yin, Z., Reuben, F., Stepney, S., & Collins, T. (2023). Deep learning's shallow gains: A comparative evaluation of algorithms for automatic music generation. *Machine Learning*, *112*(5), 1785–1822.

Appendix A: Training



(a) (b)

(c)

Figure A.1: Learning rate curves of the models trained during the grid search in Chapter 6.

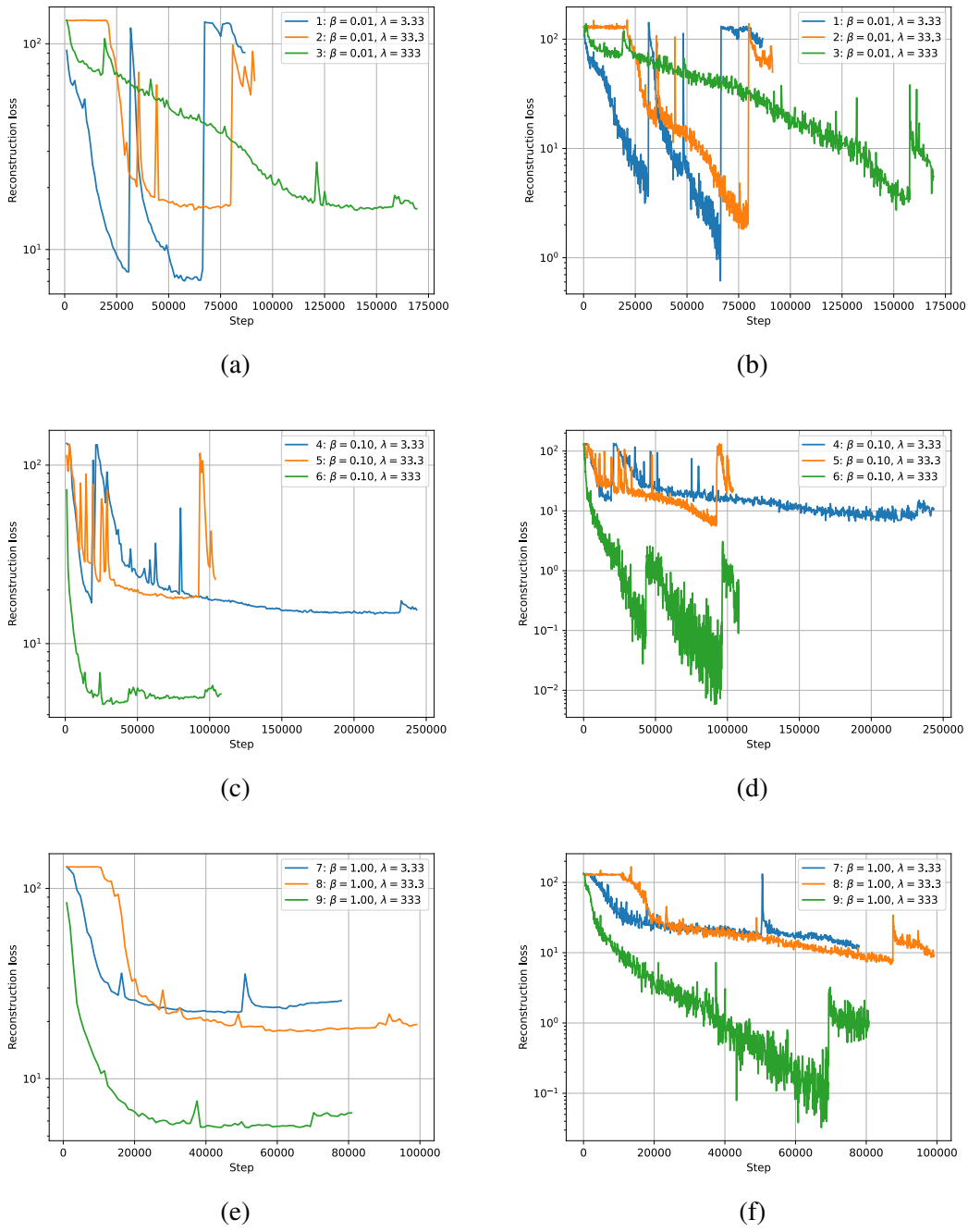


Figure A.2: Validation reconstruction loss (left) and training reconstruction loss (right) of the models trained during the grid search in Chapter 6.

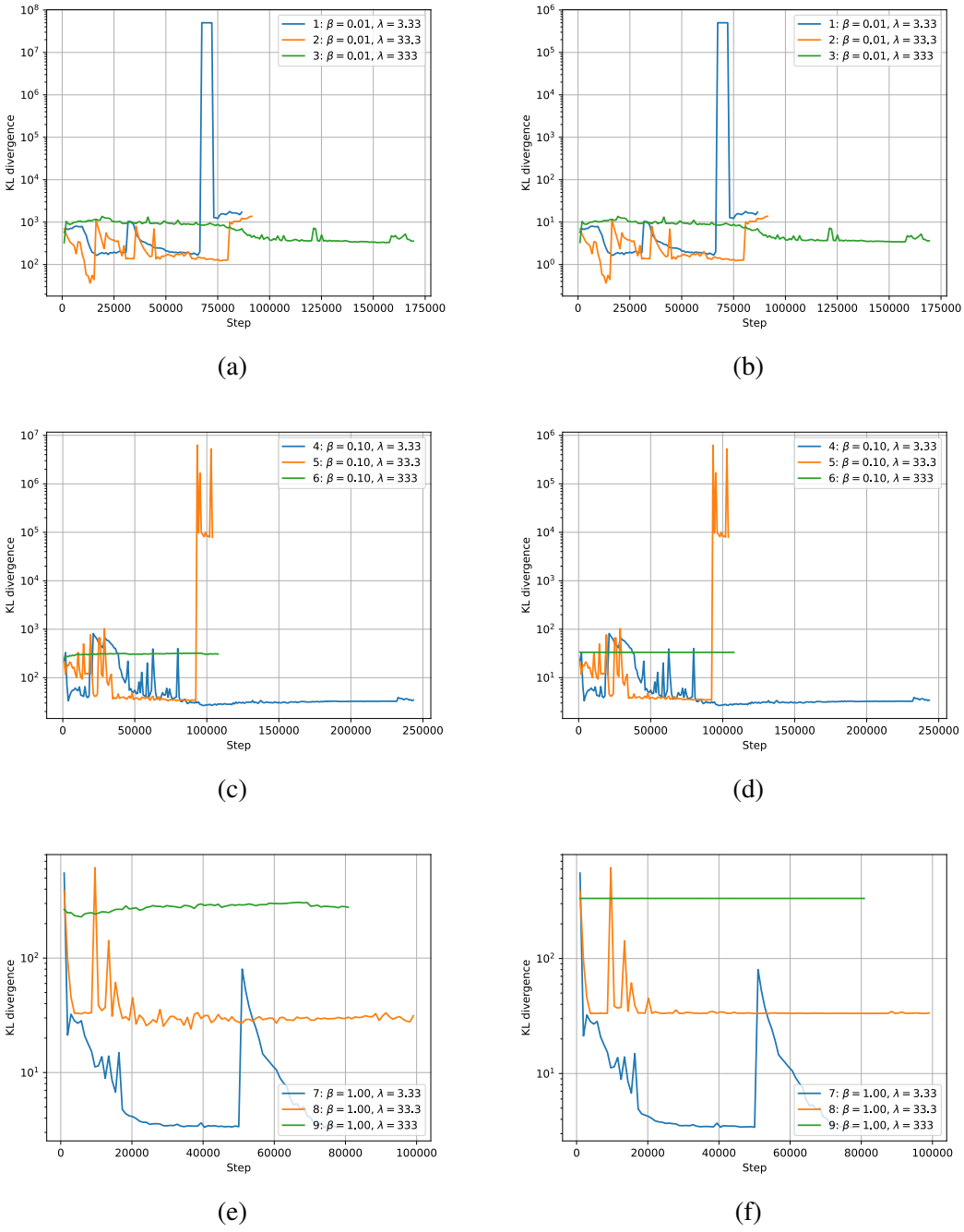


Figure A.3: Mean KL divergence per batch (left) and limited and scaled KL divergence (right) of the models trained during the grid search in Chapter 6.