Master thesis

AN ANALYSES OF THE CURRENT CAPABILITIES OF NEURAL NETWORKS TO PRODUCE MUSIC-RESEMBLING AUDIO

December 20, 2018

Sinit Tafla
VU University Amsterdam
Faculty of Sciences

Abstract

In the past years, there has been a growth in the applications of neural networks for automatic music (or audio) generation (Briot et al. (2017)). However, most of the recent research tends to be narrow-focused. In this research, we will provide a wider-range analyses of the current state of neural networks to produce music-resembling audio. We will try to reach this objective by implementing models that can analyze notes and tempo, as well as ones that can analyze raw audio waveforms. Furthermore, the models will generate music according to their finding that will be evaluated by listening to them. We will implement the waveform-based models (SampleRNN (Mehri et al. (2016)) and Wavenet (van den Oord et al. (2016c))) using two different music genres, namely: classical and techno. This allows us to examine what the influence of the type of dataset is. Finally, we will examine if lower losses also guarantee music of better quality. We found that neural networks are able to find patterns based on notes and tempo and this results in music of reasonable quality. In addition, from the two waveform-based models we implemented, SampleRNN analytically performs better and classical music was more suitable for analyses. However, all samples produced by the waveform-based models are noisy and, therefore, do not resemble music. Lastly, we could not conclude that models with a lower loss generally produce music of better quality.

Acknowledgement

I would first like to thank the company Xomnia for providing me an opportunity to do my internship and research. The company arranged a wonderful project I could work on and facilitated an office, Wi-Fi and external computer power. Special thanks go to data scientists Bart van de Poel and David Woudenberg who were my supervisors at Xomnia. They regularly advised me about the current directions and helped me whenever I was stuck. In addition, I would like to acknowledge all colleagues at Xomnia who voluntarily gave their opinion about music samples.

I would also like to thank my university supervisor professor Sandjai Bhulai, Ph.D of the Faculty of Sciences at the VU University Amsterdam. He regularly gave me feedback about my process and made sure I prioritized my work. Furthermore, I would like to mention professor Wan Fokkink, Ph.D of the Faculty of Sciences at the VU University Amsterdam. He acted as second reader and I am, therefore, thankful for his effort to evaluate my thesis.

In addition, I am profoundly grateful to Dutch DJ Joris Voorn and his team. They provided me a musical set I could analyze. Without the received files, a significant part of my research would be missing.

Finally, I want to express my profound gratitude to my family and friends who supported and encouraged me during my study and research. Without them, finishing this report would not have been easy.

Contents

1 Introduction

The generation of music through artificial intelligence has received growing attention over the last few years. Hiller & Isaacson (1959) already investigated the production of computer-generated music. Ever since, the field has made several advances. An example is the development of computer-music programming languages (e.g., Dannenberg (1997), McCartney (1996) and Boulanger (2000)), further boosting the efficiency of music creation through computers. Nevertheless, some of the most significant developments were made only recently. This is primarily caused by the applications and improvements in the use of deep neural networks (NNs) for automated music (or audio) generation, as shown by Briot et al. (2017).

Most of the recent research, however, is narrow-focused. Researchers tend to focus on a new technique (e.g., Engel et al. (2017)), a single artist (e.g., Hadjeres & Pachet (2016)) or a single type of musical dataset (e.g., Mehri et al. (2016)). The purpose of our research is to give a wider-range of insights into the usage of deep neural networks to generate music. We aim to reach this objective by comparing different models, genres and data formats. Consequently, the main research question related to this research is: "*What is the current state of deep neural networks in modelling and generating music-resembling audio?*".

This research is split in two parts, corresponding to the formats of data available, namely: MIDI and WAV. Sections 3 to 6 (Part I) discuss MIDI files and Sections 7 to 10 (Part II) discuss WAV files. We will use MIDI files to store information about musical notes and tempo, which can be translated to music. This allows us to examine temporal relationships based on a representation of music, instead of raw audio. Moreover, we will generate audio accordingly. Hence, our first sub-question is: *"Are deep neural networks capable of learning temporal relationships based on notes and tempo and generate music-resembling audio based on these relationships?"*. More specifically, we will focus on classical music samples composed by Johann Sebastian Bach. A German composer from the 17^{th} and 18^{th} century. By using this representations of music, we are saving space and require less computational power. However, we are limited by the range of sounds MIDI files can store.

On the other hand, WAV files do not suffer from this limitation. They are able to capture the characteristics of any type of raw audio. However, storing audio using WAV files requires much more data and analyzing them more complex models. A number of models have been developed to capture patterns in raw audio and generate music. Most notable is Wavenet that was developed by van den Oord et al. (2016c) (see Sections 2.1 and 8.1). Wavenet proved to successfully produce human-like music. Another notable model named SampleRNN (see Sections 2.2 and 8.2) was constructed by Mehri et al. (2016). The authors state that their own implementation of Wavenet performs worse than SampleRNN in terms of generating music. To see if this conclusion generally holds, we will compare the two models as well. The resulting second sub-question, therefore, is: *"Which state-of-the-art model is more capable of learning temporal relation-*

ships based on waveforms and generate music-resembling audio based on these relationships?".

Because WAV files can play a wide range of sounds, we can analyze different genres of music with each other as well. This provides us the opportunity to examine the influence of the type of dataset on learning capabilities. For both WAV related models, we will use two datasets: one related to classical music and the other to techno music. The classical WAV files are all pieces of Bach his "The Open Goldberg Variations" and the techno WAV files are provided by Joris Voorn, who is a Dutch DJ. The related sub-question is: *"Does the type of dataset influence the learning capabilities of models in terms of capturing patterns in music and producing music-resembling audio accordingly?"*.

Intuitively, we would think that models (or instances of models) with lower loss values produce more music-resembling audio. Nevertheless, can we expect that a difference in losses can be noticed by a human, especially if this difference is rather small? Furthermore, if a model finds a pattern that is generalizable, it will continue training using this pattern, while there is no guarantee of melodical improvements. In order to investigate the relationship between losses and quality of music samples, we state our final sub-question as following: *"Do neural networks with lower loss values also produce more music-resembling audio?"*.

Finally, we will discuss the structure of this research. Section 2 provides academic background related to the production of music using artificial intelligence. After this section, the paper will be split in two parts: Part I relates to notes and tempo analyses, and Part II relates to raw audio analyses. The sections within the two parts will have similar structures. Sections 3 and 7 will highlight the structure and preparation procedures of the MIDI and WAV files, respectively. Sections 4 and 8 discuss the methods that will be implemented. The generation, and set up of the evaluation, of the music samples will be discussed in Sections 5 and 9. The results will be given and highlighted in Sections 6 and 10. The conclusions related to both parts will be given in Section 11. In this section, we will also discuss some of the shortcomings of our research. Finally, Section 12 describes possible follow-ups to our research.

2 Related work

In this section, we will give an overview of different related studies in the field of generating music using artificial intelligence. Some of the algorithms in the studies will be further explained and implemented in Sections 4 and 8, while others will only provide scientific background.

2.1 Wavenet

Wavenet is an autoregressive probabilistic model constructed by van den Oord et al. (2016c) and is capable of predicting raw audio waveform (e.g., music) based on previous waveforms. The joint probability of a waveform $\mathbf{x} = \{x_1, ..., x_T\}$ is based on the product of conditional probabilities of preceding samples, which is summarized in Equation 1. Here, x_t represents the amplitude value of a discrete time point t.

$$p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t | x_1, ..., x_{t-1}) \tag{1}$$

The probabilities are estimated by convolutional layers. By using a softmax layer (see Section 4.1.3 for more details), the probability distribution of the categories for the next values $x_t, ..., x_T$ are estimated. Furthermore, Wavenet uses causal convolutions, which ensures that the time order is preserved and value x_t will not depend on any future time steps $x_{t+1}, x_{t+2}, ..., x_T$

Models with causal convolutions are typically faster to train, however, they require many layers or large filters to increase their receptive field. The researchers solved this issue by using dilated convolutions to increase the receptive fields, without increasing the computational costs too much. In every layer, the dilation is doubled until it reaches an output note.

During the course of their research, van den Oord et al. (2016c) performed several experiments. One of the experiments regarded the generation of novel and high-quality music. They evaluated the generated music samples subjectively by listening to the quality of the results themselves. They concluded that enlarging the receptive field was crucial to obtaining music samples that sounded human-like.

2.2 SampleRNN

Mehri et al. (2016) explore the usage of RNNs for generating waveforms. They introduce SampleRNN, a model that can perform computations at different clock-rates. This model, therefore, can allocate computational resources which results in more memory efficiency during training.

Equivalent to Wavenet, the probability of a waveform sample is calculated using Equation 1. Moreover, SampleRNN uses softmax (see Section 4.1.3) to estimate the probabilities of the future values x_{t+1},..., x_T.

Audio samples contain structures at different scales, i.e., correlations exist between neighboring samples but also between samples that are much further away. SampleRNN tackles this problem by using a hierarchy of modules. All modules consider a different temporal resolution. The lowest module in the hierarchy considers all individual samples, the higher a module is in the hierarchy, the lower its temporal resolution. Each module is conditioned by the one below it, except for the module on the bottom which produces the predictions. This enables SampleRNN to look for relationships at different time scales.

The resulting SampleRNN models are preferred by independent human raters over a number of other models, including their own implementation of Wavenet. They conclude that using a hierarchy of time scales helps overcome the problem of using RNNs to model high-resolution temporal data.

2.3 Sincet

Sincet is a model that is constructed for speech recognition. The related paper provides us valuable insights because Sincet is a neural network designed to learn patterns based on raw audio waveforms. Its purpose and design, therefore, has similarities to the waveform-based NNs we will implement (see Section 8). Ravanelli & Bengio (2018) argue that the most crucial part of wave-form based CNNs for speech recognition is the first convolutional layer. According to the authors, the importance of this layer lies in the fact it deals with high-dimensional input and is more affected by vanishing gradients. Furthermore, the filters are often noisy and take incongruous multi-band shapes that do not appeal to human intuition nor represent a speech signal well.

To cope with these shortcomings, they introduce Sincet. Unlike traditional CNNs, where the filter depends on multiple parameters, Sincet convolves the waveform with a set of parametrized sinc functions. The only parameters Sincet learns are the low and high cutoff frequencies.

The model has four main advantages over traditional CNNs for speech recognition. Firstly, Sincet mainly focuses on filter parameters that have a significant impact on the performance of the CNN. It is because of this reason that Sincet has a significantly faster convergence than other models. Secondly, Sincet drastically reduces the number of parameters in the first convolutional layer of the network. Furthermore, the number of parameters will increase at a much slower pace when increasing the number of filters or the filter length, compared to other CNNs. Finally, Sincet is more computationally efficient and the results are more human-readable and thus interpretable.

But most importantly, Ravanelli & Bengio (2018) prove that Sincet performs better in speech recognition than traditional wave-form based CNNs.

2.4 Music Style Transfer

Shuqi et al. (2018) summarize the current position of using machine learning in order to successfully transfer music. Their goal is to eliminate the underlying confusions and to highlight the findings of music transfer before the age of deep

learning. Moreover, they discuss the current limitations and future directions of music style transfer.

They argue that models have a hard time making music that is natural, creative and human-like. In addition, they state that the term "music style" is ambiguous, caused by the multi-level and multi-model characteristics of music representation. They, therefore, state a precise definition of music style transfer based on the uniqueness of music representation.

In order to clarify the aspects of music style transfer, they represent three music representations, namely: score, performance and sound representations. The first representation uses discrete features with a mix of measurement scales (e.g, sheet music notation), while the performance representations represent the interpretation of the corresponding score into motions. Lastly, sound representation is the acoustic realization of performances on a certain instrument.

Furthermore, they present 3 definitions of music style transfer: timber style transfer for score, performance style transfer for performance and composition style transfer for sound. The first type of transfer would allow us to reproduce music on different instruments while maintaining the same musical expression. Performance style transfer can be applied to transfer between interpretations of the artist of the same score representation. Composition style transfer on the other hand, refers to the variation, improvisation or re-harmonization of a piece of music.

During the course of our research, we will focus on composition style transfer. We will build models that can generate novel music based on historic time steps. The related processes will be further explained in Sections 5.1 and 9.1.

2.5 A Universal Music Translation Network

Mor et al. (2018) introduce a method for translating music across musical instruments, genres and styles. They discuss their autoencoder which can translate distorted versions of the input to undistorted versions of the output. The model learns to project input from one domain to the output from another domain.

They use a Wavenet-like encoder to transform the audio to a latent space. Next, they use a domain-specific Wavenet-like decoder to translate the audio back to the corresponding domain. This, for example, allows a musical piece to be played across different instrument.

2.6 NSynth

Engel et al. (2017) contribute to the state of generative audio modeling in two ways. First, they introduce an autoencoder that can effectively capture long-term structures without external conditioning. The resulting model can also be used for other applications, such as audio interpolation. Secondly, they introduce a general large-scale dataset called NSynth, which can be used to explore the possibilities of audio generative modeling.

Part I
Notes and tempo

3 Data

This section highlights the data we will use in order to successfully train models to generate music based on notes and tempo. As stated in Section 1, we will start by analyzing the music of Johann Sebastian Bach, a musician and composer. We will discuss the details of the data, including an explanation on how we converted the music samples to a usable format. The process of converting the music files to datasets is given in Figure 1.

We collected a total of 200 music pieces in the form of audio files that are composed by Bach[1]. The audio files are of the type MIDI (Musical Instrument Digital Interface). MIDI files differ from regular music file formats because they actually do not store any digital audio. Instead, they contain a list of instructions, which a device (e.g., a computer or cell-phone) can translate to music. MIDI files, therefore, have smaller file sizes than WAV, MP3 and other digital audio formats. Hence, we expect that training models using MIDI files will be less computationally expensive. In this research, we will partly focus on analyzing and generating music scores in the form of musical notes. It is important to define an adequate representation to capture the complexity of audio. To reach this objective, we will use three objects to represent the music of Bach, namely: notes, chords and offsets.

MIDI objects contain information about notes being played at a point in time, namely: pitch, octave and offset. The pitch describes the frequency of a note, i.e., how high or low a note is. A pitch is represented by the first 8 letters of the alphabet, where A and G represent the highest and lowest sounds, respectively. A specific pitch may contain different variations whose frequencies equal the multiples of the original pitch frequency. Such variations of a pitch are called octaves. Lastly, the offset is the point in time a note object is being played, starting from the beginning of the audio file.

However, notes can be played simultaneously. This is where chords objects play their role. Chord objects are basically lists of notes that are being played at the same time.

The MIDI objects consist of a note or chord and their corresponding offset. Before we can analyze these objects, we have to convert them to a format that our neural networks can understand. The notes will be denoted by the Scientific Pitch Notation (SPN), which summarizes the note its octave and pitch. An example is "A3", where "A" represents a specific pitch and "3" corresponds to the octave. Chords on the other hand, will be written in the well-known normal order. The format summarizes the list into a single string where the notes are represented by integers and are separated by a dot. For example, if we would apply normal order to a chord containing "B-4", "D5" and "F5" in its list, the

[1]The music of Bach is retrieved from "http://www.bachcentral.com/."

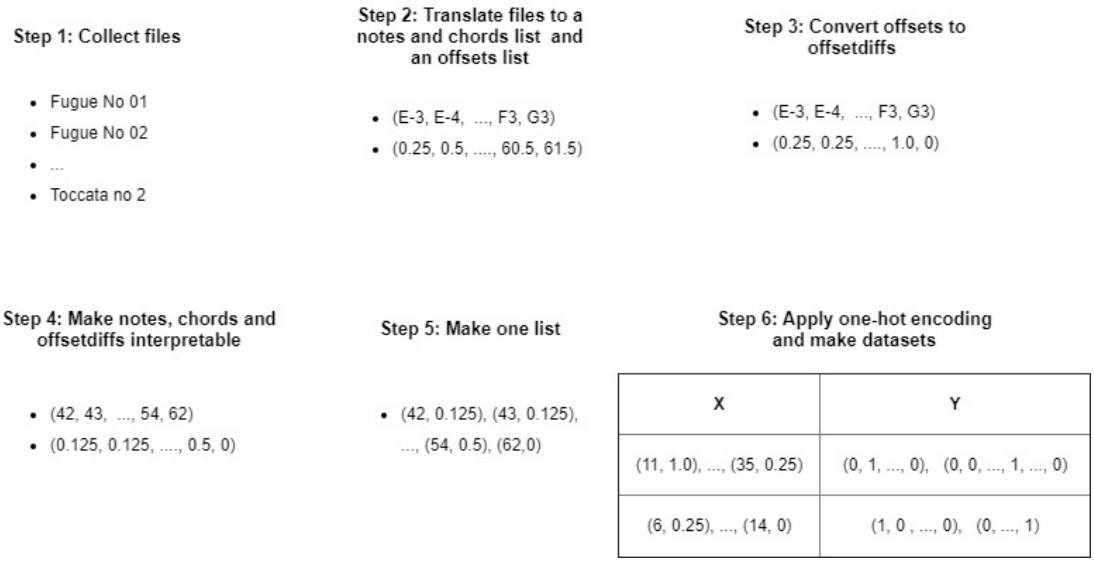

Figure 1: Data preparation steps for MIDI files

resulting string would be "10.2.5". Moreover, the offset is expressed as a float that represents the location of the note or chord on the piece. By using these representations for music, we save computational power and thus make it easier for our NNs to analyze audio. If we would use raw audio to represent a note or chord, we would need tens, if not hundreds, of thousands of objects. If we would store a note lasting 2 seconds on CD (where the sample rate equals 44.1 kHz), for example, we would need $44.1 \cdot 1000 \cdot 2 = 88,100$ values. Whereas we only need the SPN and the offset to reproduce the same note using MIDI files.

Additionally, the audio files may contain several instruments. This adds extra complexity because we also would have to analyze which sequences of objects correspond with a specific instrument. Furthermore, the purpose of our research in this part is to generate new music, by predicting new notes and chords. We are not interested in the set of instruments they should be played on (see Section 1). Consequently, we only consider the notes and chords that belong to a single instrument, namely a piano. This allows us to start off simple and still be able to analyze music extensively. We build our dataset by starting with empty lists. For each MIDI file, we eliminate the notes, chords and offsets belonging to other instruments. The remaining chords and notes objects are added to one list and the offsets to another. The resulting length of both lists equals 134,306 (L).

However, we are not interested in the point in time a note or chord is played, but rather in the difference of time between two succeeding objects. We introduce a new object called offsetdiff (od), representing the time difference between two succeeding offsets. The related values of the offsetdiffs are stored in a dif-

ferent list. The values are determined by Equation 2. Since the offsetdiff of the last object can not be calculated, it is set to 0. Using offsetdiffs allows us to analyze and produce music of different tempos. Additionally, we allow the models to find correlations between the notes and tempo. Consequently, we have more information available which we expect to lead to music of better quality.

$$od_t = \begin{cases} o_{t+1} - o_t & t < L \\ 0 & t = L \end{cases} \tag{2}$$

Not all note, chord and offsetdiff values are presented a sufficient number of times for a model to find key characteristics. Hence, we replace all objects which occur fewer than 100 times. Notes and chords will be replaced by one of the notes or chords objects that occur a 100 times or more and the same logic holds for the offsetdiff values. The sample probability is proportional to the underlying ratio of the objects that occur a sufficient number of times. This procedure reduces the number of distinct note and chord values (K_{no}) from 219 to 65 and the number of distinct offsetdiff values (K_{of}) from 79 to 12. However, the note, chord and offsetdiff values are not interpretable for a neural network yet and need to be converted to values between 0 and 1 ultimately. To reach this objective, we first convert the values of the notes and chords to integers as an intermediate step. We assign the distinct values to an integer between 1 and K_{no}. All values in the note and chord list are replaced with their corresponding integer as a result. The notes and chords values will be further manipulated by embedding layers (see Section 4.1.2). Similarly, we have to transform the values of the offsetdiffs. We transform the values by using min-max normalization. The corresponding formula is given in Equation 3, where the values to normalize are represented as $x = \{x_1, x_2, .., x_n\}$ and the resulting value of point i as z_i.

$$z_i = \frac{x_i - min(x)}{max(x) - min(x)} \tag{3}$$

Next, we will combine the two lists in one. As a result, every time step has two features: a note or chord and an offsetdiff.

The next step is to determine the set of time steps $\mathbf{X}$ corresponding to target values $\mathbf{Y}$. The sequence length S determines the number of preceding elements our neural networks consider. Every row in the dataset consists of a target value y_j and its S historical time steps. Here x_j^1 represents the value at time point $t - S$, x_j^2 the value at time point $t - (S - 1)$, and so on. The number of rows N in the dataset will equal $N = N_m - S$, where N_m represents the total number of time steps. This is because the first S values will not be considered, due to the fact they lack historical values. Because the objects of all music files are appended to the same list, the time steps of a number of rows contain overlap between files. These rows in question are, therefore, deleted.

Next, we convert the target values $\mathbf{Y}$ to two arrays of booleans where the lengths equal K_{no} and K_{of}, respectively. For each row y_j, exactly one value in both sets equals "1". This process is also called one-hot encoding.

Furthermore, to judge the performance of the models as objectively as possible, we split the dataset in a training, test and validation set. We randomly sample 80% as training and validation set and use the remaining 20% as test set. Moreover, within the training and validation set, we similarly sample 80% as training set and assign the remaining 20% to the validation set.

4 Methods

This section will highlight the different methods we will implement in order to analyze the capability of learning and reproducing the music of Bach (based on notes and tempo). The models that will be used for the MIDI files will generally be less complex, since they will not cover the same amount of data. In this section, we will highlight the structure of each model and explain why we make relevant choices.

4.1 LSTM models

The first type of models we will implement are LSTM (Long Short-Term Memory) models. They are a type of recurrent neural networks (RNNs) that were first introduced by Hochreiter & Schmidhuber (1997). The main advantage of LSTM models is that they can handle long-term dependencies better. Chung et al. (2014) already concluded that LSTM models work significantly better in tasks as modeling polyphonic music and speech signal modeling. Normally, the gradient of the loss functions of RNNs decays exponentially over time, which is also known as the vanishing gradient problem. LSTM networks solve this problem by having a memory cell that can contain more information over longer periods of time.

Each unit has three gates, namely: a forget, input and output gate. Each of these gates has their own task. Intuitively, input gates control to what extent new information is passed to the cell and forget gates determine to what extent values remain in the cell. The output gates determine to what extent the value in the unit is used to calculate the new activation function. The values of the input (i_t), output (o_t) and forget gate (f_t) at time point t are calculated using the formulas given in Equations 4, 5 and 6, respectively.

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \tag{4}$$

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \tag{5}$$

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \tag{6}$$

where x_t denotes the input at time point t, h_t the output vector of the LSTM unit and b_q the bias. Furthermore, W_q and U_q correspond to the weights of the input and recurrent connections, respectively. It is important to note that q represents the related gate, or in other words: $q \in \{i, o, f\}$.

In addition, each unit contains a memory cell c_t, which is used to store values for either long or short periods. The value of c_t is determined by Equation 7. After computing c_t, we can determine the new value of the output h_t by using Equation 8. Here $\circ$ represents the element-wise product.

$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \tag{7}$$

$$h_t = o_t \circ \sigma_h(c_t) \tag{8}$$

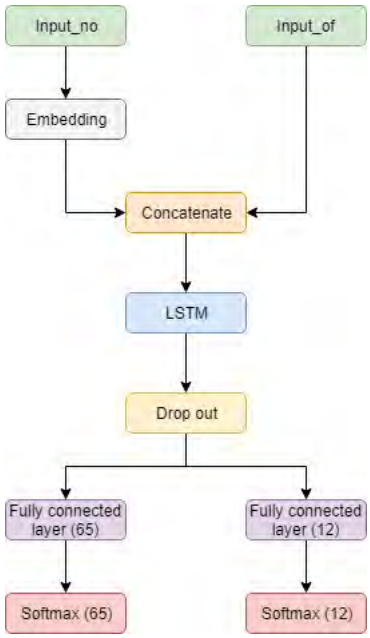

Figure 2: Design LSTM-I

4.1.1 Design LSTM models

To test the ability of finding patterns in MIDI files, we will implement two different LSTM models. We will refer to these models as LSTM-I and LSTM-II. Their corresponding designs are given in Figure 2 and 3, respectively. Both models have an LSTM layer with an output dimension of 256. The main difference between these models is the fact that LSTM-II has an additional LSTM layer with 512 neurons.

4.1.2 Embedding and concatenate layer

The notes and chords are described as integer values (see Section 3) that are meaningless in this representation. To transform these integers to values that are more preferable for NNs, we will use an embedding layer. First, the two features are split for every time step. The resulting notes and chords and offsetdiffs will be called *input_no* and *input_of*, respectively. The embedding layer will only be applied to *input_no*. In their essence, embedding layers transform integers to dense vectors with chosen size e_l, eliminating the natural ordering of integers. Each note and chord will be assigned to exactly one vector. All vectors are updated while training the NN in question. After training, the integers can be mapped to points in multi-dimensional space using their corresponding vectors. This allows us to visualize the relationships the model finds between notes and chords. A model will consider two points to be more similar if their distance

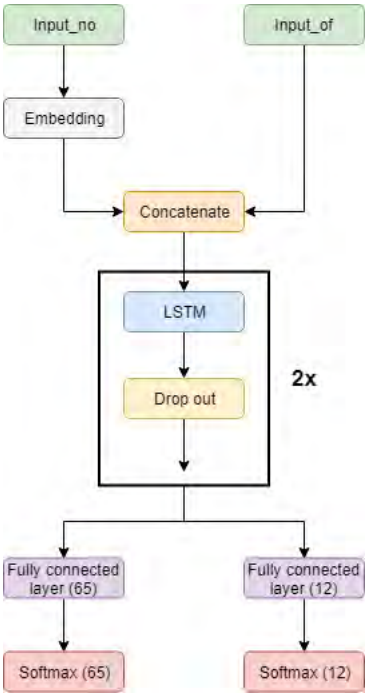

Figure 3: Design LSTM-II

is smaller. By analyzing the resulting embedding matrix (the matrix of the embedding vectors), we are able to see if the model its findings are in line with music theory (see Section 6). This describes the set of rules that explain the relationships between the notes we hear in musical pieces.

The results of the embedding layer and *input_of* need to be combined together so all related values can be considered by an LSTM. This is where a concatenate layer plays its role. For every time step, it will concatenate both sets of values. This results in a feature set of length $e_l + 1$ for every time step.

4.1.3 Other layers

Two fully connected layers will serve as the second last layers in our RNN models. They will convert the current dimension of the values to K_{no} and K_{of}, respectively. This is a necessary step in predicting the probabilities of observing a note, chord and offsetdiff.

Moreover, softmax layers will follow to determine the estimated probability of observing a certain category. The corresponding formula is given in Equation 9.

$$\sigma(z)_j = \frac{e^{z_j/T}}{\sum_{k=1}^{K} e^{z_k/T}} \tag{9}$$

where z represents the output vector of the fully connected layer and $\sigma(z)_j$ the probability of observing j given z. The temperature T ($0 \leq T \leq 1$) controls to what extent randomness can occur in predictions. A higher temperature results in more diversity, but also more mistakes during sampling (see Section 5.1). A low temperature on the other hand, results in more confident and conservative predictions. However, we do not vary the temperature during the comparisons of models because we do not know what effect randomness has on the quality of the resulting music samples. Consequently, we set $T = 1$ to calculate the errors in Section 6. Moreover, one of the softmax layers determines the probability of observing a note or chord, while the other calculates the probability of observing an offsetdiff.

Furthermore, we will use Adam optimizer (Kingma & Ba, 2014) to update the weights to minimize the categorical cross entropy loss (given in Equation 10) that estimate the errors in our network. Here C represents the number of different categories and N the number of observations. y_j^c equals "1" if the target value of row j belongs to category c and "0" otherwise. Lastly, $\hat{y_j^c}$ represents the estimated probability of the target value of row j belonging to category c.

$$-\frac{1}{N}\sum_{j=1}^{N}\sum_{c=1}^{C} y_j^c \cdot log(\hat{y_j^c}) \tag{10}$$

Categorical cross entropy will use both softmax functions to generate two individual loss values, implying how well the model estimates notes and chords

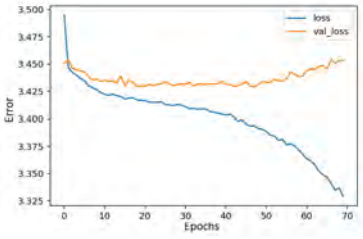

Figure 4: Before regularization

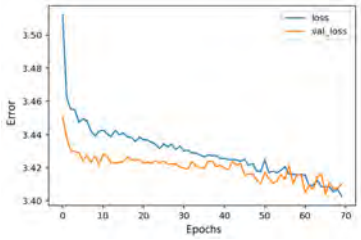

Figure 5: After regularization

and offsetdiffs. Different instances will be compared to each other by taking the mean of the two resulting losses.

4.1.4 Preventing overfitting

To prevent overfitting, we will implement several methods. First, we will insert a dropout layer after an LSTM layer (see Figures 2 and 3). During training, the weights of a vector h_t will be set to zero with a probability of $1 - p_{dr}$. At each training stage, a different set of neurons will be used to calculate the output. By randomly eliminating neurons during training, we can reduce co-dependence between neurons, which ultimately leads to less overfitting. Secondly, we will extend our loss function with L2 regularization to penalize large weights. This prevents the parameters from becoming excessively large and dominating the neural network. The formula of L2 regularization is given in Equation 11, where λ_{l2} controls the degree to which large weights are being penalized.

$$\lambda_{l2} \sum_{i=1}^{n} \theta_i^2 \tag{11}$$

To test the effect of regularization, we will train two instances of LSTM-I using an identical arbitrary parameter set. One of the models will use regularization while the other will not. Next, we plot the losses of their train and validation sets against the epochs. The results are given in Figures 4 and 5, where "loss" and "val_loss" represents the error of the training and validation set, respectively. We can clearly observe that regularization has an effect on overfitting. In Figure 4, we can see overfitting occurs around 30-35 epochs, while overfitting does not seem to occur in Figure 5. Furthermore, L2 regularization seems to not negatively affect the training of the model. Both losses are still decreasing after 70 epochs in Figure 5.

Thirdly, we will implement early stopping, a method that stops training as soon as the loss of the validation set does not improve for a number of epochs. This value is also known as the patience.

4.1.5 Adaptive learning

In addition to preventing the LSTM models of overfitting, we also want the optimizer to explore the search space as much as possible in the beginning and gradually consider points closer as time passes. We can establish this objective by decreasing the learning rate of the Adam optimizer during training. More specifically, we start with a learning rate and proportionally decrease the value over the epochs. Hence, the learning rate lr_t at time point t will follow Equation 12, where $1 \leq t \leq T$.

$$lr_t = lr_0 \cdot \frac{T - (t-1)}{T} \tag{12}$$

4.1.6 Parameter optimization

Finally, we want to optimize the parameters. Because training a single model is computationally expensive, techniques as evolutionary algorithms or simulated annealing are less preferred. Hence, we will use a grid search to optimize the parameters. We will select a number of parameter values and calculate the losses for the training and validation set. For each parameter set, we will run LSTM-I and LSTM-II for 200 epochs, that is if early stopping does not end the process beforehand. The patience will be set to 10. Moreover, the optimal parameter set will be equivalent to the one that minimizes the loss of the validation set. The sequence length S will be set to 60 and the output dimension of the embedding layer e_l to 3. The parameters we will optimize are: the fraction of neurons that will pass through the drop out layer $p_{dr} \in \{0.1, 0.3, 0.5, 0.7\}$, the regularization term $\lambda_{l2} \in \{0.005, 0.001, 0.0001\}$ and the initial learning rate of the Adam optimizer $lr_0 \in \{0.007, 0.005, 0.003\}$.

4.2 CNN

The second type of model we will consider to replicate music using MIDI files is convolutional neural network (CNN) and was introduced by LeCunn & Yann (2013). More specifically, we will implement a 1-dimensional CNN. CNNs use multilayer perceptrons designed to require minimal preprocessing . Hence, we expect to train CNNs much faster than RNNs (e.g., LSTMs).

CNNs are characterized by their convolutional layers. Such a layer applies one or more convolutional operators to the input and passes the results on to the next layer. By using filters with defined weights that slide over the input, matrix manipulations are performed and the resulting sum is mapped into a feature map. This process is summarized in Figure 6. The image shows a filter with the values 1, 3 and 1 for the weights w_1, w_2 and w_3, respectively. The receptive field of the filter, i.e., the area the filter considers, is 3. We can apply multiple filters to the same input, resulting in multiple feature maps. The step size by which a filter moves is called a stride and equals 1 in Figure 6.

Notice that the resulting feature map has fewer values, causing information to be lost. We will ensure maintaining valuable information by implementing

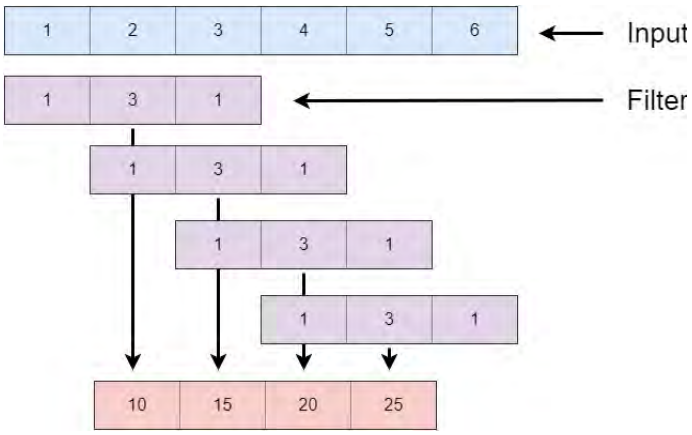

Figure 6: Example of CNN filters

zero padding. On each side of an input vector x_j a value of "0" is added. This ensures that the resulting feature maps have the same number of values.

4.2.1 Structure CNN

The design of our CNN is given in Figure 7. Again, the input is split in *input_no* and $input\_of$, where *input_no* passes through an embedding layer (see Section 4.1.2). The neural network we will implement contains three 1-dimensional convolutional layers. The input will pass through three subsequent combinations of a 1-D convolutional layer and a drop out layer. The first, second and third convolutional layer use 128, 64 and 32 filters, respectively. Note that the output dimension after the third drop out layer is $S \times 32$. In order to convert the findings of the model to probabilities, the matrix must be converted to a single vector. This is where the flatten layer plays its role. The layer converts the current output dimension to a vector of length $S \cdot 32$. To add extra complexity to the model, we add a fully connected layer with 256 neurons. This makes it possible to learn more intricate patterns. Like explained in Section 4.1.3, we connect the output to two additional fully connected layers, one with K_{no} neurons and the other with K_{of} neurons. Finally, they are both followed by their own softmax activation function.

4.2.2 Training and parameter optimization

Similarly to Section 4.1, we will implement early stopping with a patience of 10 and use L2 regularization in order to prevent overfitting. Moreover, we will use categorical cross entropy to calculate the loss and an Adam optimizer. The learning rate of the optimizer will again be decreased proportionally over time to effectively utilize the search space (see Section 4.1.5). Lastly, the stride will be set to 1.

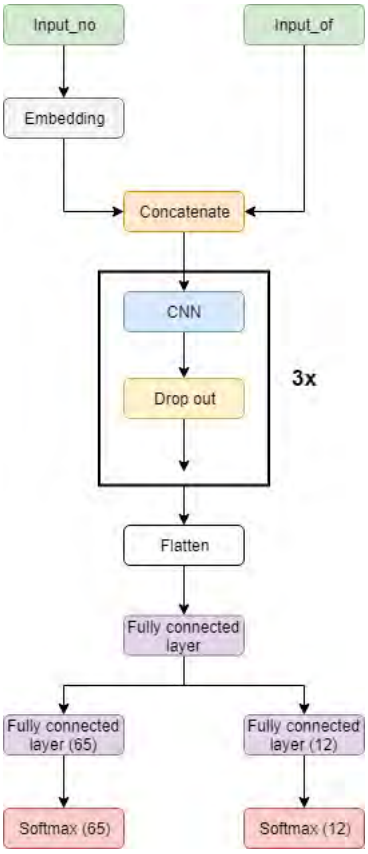

Figure 7: Design CNN

Next, we will discuss how we will optimize the parameters. The sequence length S will again be set to 60 and e_l to 3. Also, we will optimize four parameters using a grid search. The first three parameters are equivalent to ones we optimize for the LSTM networks, namely: $p_{dr} \in \{0.1, 0.3, 0.5, 0.7\}$, $\lambda_{l2} \in \{0.005, 0.001, 0.0001\}$ and $lr_0 \in \{0.007, 0.005, 0.003\}$. The last parameter is the filter length $f \in \{2, 3, 4, 5\}$. We will not optimize the filter lengths of the three convolutional layers independently from each other. Instead, we will use the same value for f in every run for the three convolutional layers. Adding extra parameters to a grid search increases the number of runs exponentially. Thus, by optimizing just one value of f for the three convolutional layers, we are saving a tremendous amount of computational cost.

4.3 Markov chains

The last type of model we will implement for the MIDI files is Markov model. A Markov model is a model that satisfies the Markov process that was introduced by Markov (1906). However, we do not expect this model to perform well, nor to generate qualitative music. Because of the simplicity of this model, it will primarily serve as baseline, i.e, the other models should at least perform better than the Markov model. Yanchenko (2017) concluded that Hidden Markov models (HMMs) are limited in reproducing music-resembling audio. For this reason and the sake of simplicity, we will implement a Markov chain instead of an HMM. Models with a higher value for the loss on the test set than the baseline are said not to learn any significant patterns, not even the most simple ones. Models with a similar value for the loss, however, only learn the most straightforward relationships. Or in other words: the ones that are simple enough for the Markov chain to find as well.

The most important concept of Markov chains is the Markov property. This property says that the probability of future states solely depends on the current state. This idea is summarized in Equation 13, where $X_1, ..., X_n, X_{n+1}$ represent a sequence of states.

$$P(X_{n+1} = x_{n+1} | X_1 = x_1, ..., X_n = x_n) = P(X_{n+1} = x_{n+1} | X_n = x_n) \quad (13)$$

Because we have two features in every time step, we will implement two separate Markov chains. One for the notes and chords and the other for the offsetdiffs. In both cases the estimated probabilities will be based on both the previous note or chord and the previous offsetdiff. For both models, we will only construct a single version. Since, we will not perform any parameter optimization, making a validation set will be unnecessary. The training set will contain 80% of the data and the test set 20% of the data. Next, we will construct the transition matrices for both models. The number of distinct combination of notes and chords and offsetdiffs in the training set (and thus the number possible current states) equals 635. On the other hand, the number of future states will equal K_{no} and K_{of} for the Markov chains of the notes and chords and offsetdiffs,

respectively. The resulting two matrices will, therefore, be of size $635 \times K_{no}$ and $635 \times K_{of}$. For every combination of notes or chords and offsetdiffs, we will count the occurrences of the succeeding objects in the training set. This result in arrays of integers of sizes K_{no} and K_{of} for every current state. Often in these resulting arrays, counts of 0 are present. However, intuitively we would expect some probability of a note, chord or offsetdiff to follow after a current state. We, therefore, will prevent counts of 0 by using Equation 14.

$$\hat{\theta_j} = x_j + \alpha \tag{14}$$

where x_j is an array of occurrences and α will be set to 1. To convert the counts to probabilities, we will use equation 15, where $1 \leq c \leq C$.

$$\hat{y_j^c} = \frac{\hat{\theta_j^c}}{\sum_{k=1}^{C} \hat{\theta_j^k}} \tag{15}$$

Again, we will use categorical cross entropy to calculate the loss. For every row in the test set, we will check if the combination of a note or chord and a offsetdiff is present in the training set. If the combination is present, we can estimate the probabilities for both the succeeding notes and chords and offsetdiffs using Equation 15. If the combination is not present, the estimated probabilities for the note and chords will be set to $\frac{1}{K_{no}}$ and the estimated probabilities for the offsetdiffs to $\frac{1}{K_{of}}$.

5 Experiments

Next to having analytically satisfying results (i.e., a low error), we also aim to produce audio that is sonically of quality. However, the quality of music and sound in general is typically a subjective measurement. In this section, we will highlight the different approaches we will take in order to give a judgment that is as objective as possible.

5.1 Sampling music

Before we conduct our experiments on music quality, we have to sample music from our models.

We start by randomly selecting an input vector v_0 as starting point from the related test set. A model will then use v_0 to estimate the probabilities for all categories of the notes and chords and offsetdiffs. The output for LSTM-I, LSTM-II and CNN is given by softmax layers that use Equation 9. The next step is to select a value for the temperature T. A low value for T results in conservative predictions. A possible consequence is that only a select number of notes, chords and offsetdiffs will be be heard. A high value for T creates more differences in results, which one can perceive as creativity. Although more variety of notes, chords and offsetdiffs could lead to more randomness and thus music of less quality. We select the optimal temperature by sampling music using different values for T and evaluating them ourselves. We conclude that the best quality arises when we make a balance between the two extremes and set $T = 0.6$ during sampling. Note that this is the only time where we vary T, during parameter optimization and training T is set to 1. The probabilities of the Markov chain model, however, are estimated using Equation 15. Based on the resulting probabilities, $\hat{w}_{t,d}$ is determined. This represents the estimated category value for time point t. Here, d relates to the specific feature. $d = nc$ when we refer to the notes and chords and $d = of$ when we refer to the offsetdiffs.

For every output file, we start by randomly selecting a vector v_0 of length S from our test set. Where $S = 1$ for the Markov chain model. Moreover, we start with an empty output list and an offset of 0.0. Based on v_0, we will estimate the probabilities $\{\hat{y}_t^c : 1 \leq c \leq K_{no}\}$ and $\{\hat{z}_t^c : 1 \leq c \leq K_{of}\}$, for the notes and chords and the offsetdiffs, respectively. The values for $\hat{w}_{0,nc}$ and $\hat{w}_{0,of}$ are sampled according to the ratio of the estimated probabilities. A note or chord is constructed by converting its corresponding estimated integer $\hat{w}_{0,nc}$. To determine the related offsetdiff, we first convert $\hat{w}_{0,of}$ to the correct offsetdiff. The result is added to the current offset value. Finally, both the estimated note or chord and the new offset value are appended to the output list. Furthermore, $\hat{w}_{0,of}$ is transformed and normalized like we did for all offsetdiffs values (see Section 3). The integer value of $\hat{w}_{0,nc}$ stays in its original form. A new input vector is created, where $v_{t+1}^{l-1} = v_t^l$ (for $l = 2, ..., S$) and v_{t+1}^S equals the resulting value of $\hat{w}_{0,nc}$ and the transformation of $\hat{w}_{0,of}$. Finally, the list of new objects is converted to a MIDI file.

5.2 Listening session I

In order to answer the research questions stated in Section 1, we need to measure the quality of the music samples. A group of volunteers will assist us by giving their judgment about a set of generated audio files. In this section, we will describe the set-up of two experiments. The first experiment relates the sub-question: "*Are deep neural networks capable of learning temporal relationships based on notes and tempo and generate music-resembling audio based on these relationships?*".

We will sample audio files of 10 seconds. I ($I = 10$) samples will be generated by randomly picking one of the MIDI NNs (LSTM-I, LSTM-II or CNN) to produce a specific music file. Each of these samples will be paired with a sample produced by the Markov chain. The files provided by the MIDI NNs and the Markov chain will be referred to as set A and B, respectively. First, we will see if we can make obvious conclusions ourselves. If the samples in set B, for example, only contain one note, it would be unnecessary to continue. It is important to be precautions with our judgment and to only cancel the experiments when conclusions can be drawn extremely easily. Otherwise we will proceed using the opinions of our volunteers. For every pair, the volunteers will be asked which file sounds more like music or whether they do not have a preference. This way, we can see if neural networks can find more complex patterns in music and these findings are noticeable for a human ear. We introduce a preference score PS_j^W, representing the preference of a volunteer j ($1 \leq j \leq J$) towards set W. The higher the score, the more the samples of W are preferred. This value is based on the comparisons between the counterparts of the individual pairs. The result of a single comparison i ($1 \leq i \leq I$) of volunteer j is summarized by $x_{i,j}^A$ and $x_{i,j}^B$. The two variables describe the performance of the counterparts from sets A and B, respectively. Equations 16 and 17 determine their values. The results over all pairs are used by Equation 18 to calculate PS_j^A. The definitions of $x_{i,j}^A$ and $x_{i,j}^B$ cause PS_j^A to be closer to 0.5 when there is little noticeable difference for a listener. On the other hand, when there is a significant difference in perceived quality, the preference score will further away from 0.5.

$$x_{i,j}^A = \begin{cases} 1 & \text{Volunteer } j \text{ prefers sample } i \text{ from A} \\ 0 & \text{Volunteer } j \text{ prefers sample } i \text{ from B} \\ 1 & \text{Volunteer } j \text{ has no preference} \end{cases} \tag{16}$$

$$x_{i,j}^B = \begin{cases} 0 & \text{Volunteer } j \text{ prefers sample } i \text{ from A} \\ 1 & \text{Volunteer } j \text{ prefers sample } i \text{ from B} \\ 1 & \text{Volunteer } j \text{ has no preference} \end{cases} \tag{17}$$

$$PS_j^A = \sum_{i=1}^{I} \frac{x_{i,j}^A}{x_{i,j}^A + x_{i,j}^B} \tag{18}$$

Furthermore, we will ask the listener to rate both sets of audio (the MIDI NN set and the Markov chain set) using Absolute Category Ranking (ACR). Where the levels of the scale (sorted by quality in increasing order) are: bad, poor, fair,

good, excellent. The ACR score can be evaluated by assigning numbers 1 to 5 to the levels, where 1 means bad, 2 means poor, etc. This allows us to judge how well the audio sets resemble music. Additionally, we will ask a volunteer why they give a certain ACR score. By analyzing these answers, we can provide insights into important characteristics of music-resembling audio. Thus, we can give a judgment about what deep neural networks should be looking for.

Next, we would like to see if the MIDI NN with the lowest test loss also performs significantly better in producing music. Thus, we would like to answer the sub-question: *"Do neural networks with lower loss values also produce more music-resembling audio?"*. To reach this objective, we will have a similar set-up as described above (compose pairs, see if experiment is necessary, ask ACR score, etc.). The volunteers and the music sample sets, however, will be different. For the second experiments, 10 samples will be generated using the best performing MIDI NN. Each of these samples will be paired with an audio file produced by one of the other two NNs.

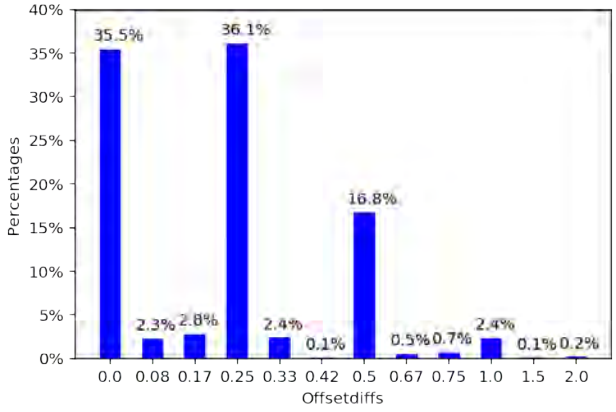

Figure 8: Distribution of the offsetdiffs in the training set

6 Results & Evaluation

In this section, we will compare the results of the different models. First, we will compare the analytical performance of the models to each other by looking at their abilities to categorize objects and the relevant loss values. After judging their learning capabilities, we will analyze the quality of the produced music.

6.1 The analytical evaluation set-up

For every NN, we will first discuss the chosen parameters and the loss development. After the model instance with the lowest loss is determined, the loss of the test set will be calculated.

Furthermore, we will test how well a model is capable of categorizing offsetdiffs. To test the performance, we will construct confusion matrices. This type of visualization allows us to summarize the category-level performance. Each column represents a single possible prediction of our model. The rows represent the actual value corresponding to that prediction. The values are normalized, i.e., the sum of each row equals 1. A model with high values on the diagonal is said to perform adequately. Low values on or high values outside the diagonal means the model has trouble predicting certain offsetdiffs. Furthermore, we will compare confusion matrices with Figure 8, that represents the distribution of the offsetdiffs in the training set. More specifically, we will examine if a model has trouble learning offsetdiffs with little occurrences.

Moreover, we also want to test a model's capability of categorizing notes and chords. In contrary to the offsetdiffs, confusion matrices will not provide us much insight, because the number of distinct categories ($K_{no} = 65$) is too high. This causes the confusion matrix of becoming too large and unclear. Instead, we will use a different approach. The objective is to conclude if the model is

capable of finding patterns between notes that are in line with music theory[2] (the set of rules that explain the relationships between notes). We will reach this by making 3D plots of the parameters of the embedding layers. In these plots, every axis represents a parameter value. Notice that $e_l = 3$ (see Sections 4.1.6 and 4.2.2), hence, the usage of 3D plots.

In the dataset, we have 12 notes, namely: C, C#, D, E-, E, F, F#, G, G#, A and B-. It is important to mention that '#' represents a sharp and '-' a flat. By multiplying the frequency of one note by 1.0595, the frequency value of the next note will be derived. For example, if we would multiply the frequency of C, we would get the value $261.6 \cdot 1.0595 = 277.2$Hz. This value equals the frequency of C#. Performing the same multiplication would result in the frequency of D and so on. An important notion is that $\sqrt[12]{2} = 1.0595$. That means that if we would complete a round of 12 multiplications starting with C, we would end up multiplying the original frequency value of C by 2. Or in other words, with C one octave above, denoted by the last number of a note. For example: E-4 is one octave above E-3, B4 one octave above B3 etc. Thus, relationships exist between notes in the same octave (i.e., ending with the same integer) and between the same notes in different (e.g., F3 and F4). In addition, we will also examine if there is a clear distinction between notes and chords.

We can determine how similar a model finds two notes or chords by looking at their distance in the related 3D plot. If the corresponding Euclidean distance is lower, the points are more similar to each other. The Euclidean distance can be described as the straight-line distance between two points in Euclidean space. The corresponding formula is given Equation 19, where p and q represent 2 points in Euclidean n-space.

$$d(p, q) = d(q, p) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_n - q_n)^2} \tag{19}$$

6.2 Results LSTM-I

We start by discussing the results of LSTM-I. After performing parameter optimization (see Section 4.1.6), we set $p_{dr} = 0.7$, $\lambda_{l2} = 0.001$ and $lr_0 = 0.003$. The development of the losses of the validation and training set are summarized in Figure 9. The training is stopped prematurely by early stopping because the loss of the validation set stops decreasing.

6.2.1 Performance offsetdiffs

The confusion matrix of LSTM-I is given in Figure 12. We can observe that the model is only capable of predicting a few offsetdiffs well, namely: 0.0, 0.25, 0.3333 and 0.5. However, it seems that this is caused by the fact that LSTM-I is prone to predict one of the four values, as their corresponding columns contain high values. From Figure 8, we can observe that these values occur a significant number of time (except for $od = 0.3333$). In addition, we see that the model

[2]Information about music theory retrieved from "http://www.simplifyingtheory.com"

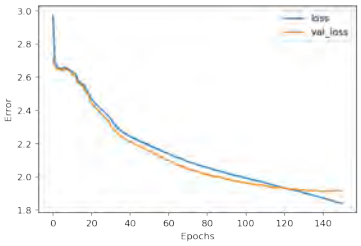

Figure 9: Training results LSTM-I

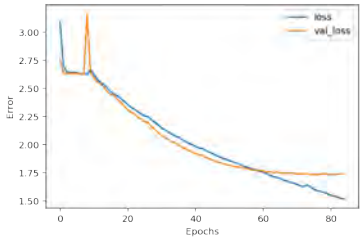

Figure 10: Training results LSTM-II

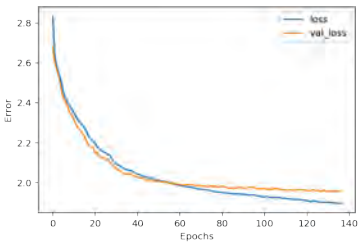

Figure 11: Training results CNN

almost never predicts offsetdiff values that occur less than 2% of the time in our training set. These values are: $od \in \{0.4167, 0.6667, 0.75, 1.5, 2.0\}$.

6.2.2 Performance notes and chords

The weights of LSTM-I's embedding layer are shown Figure 13. We can observe that LSTM-I is capable of matching notes with similar frequencies together. Notes within the same octave are grouped together. For example, notes ending with '3' are grouped together on the bottom and notes ending with '4' are grouped at the left side. Moreover, we can observe some individual relationships. An example is G3 and G#3, where the difference of frequencies is only a factor of 1.0595. However, the model does not seem to group the same notes in different octaves together. F3 and F4 are pretty far away, for example. Furthermore, all chords are clustered in a dense area, meaning the model sees a clear distinction between notes and chords.

6.3 Results LSTM-II

Secondly, we will analyze the results of LSTM-II. The resulting parameters after optimization are: $p_{dr} = 0.7$, $\lambda_{l2} = 0.0001$ and $lr_0 = 0.005$. The losses of the validation and training set are plotted against the epochs in Figure 10. We can observe that the model has trouble learning at first, but starts learning faster after 10 epochs. Furthermore, we can see that the model overfits fast, as early

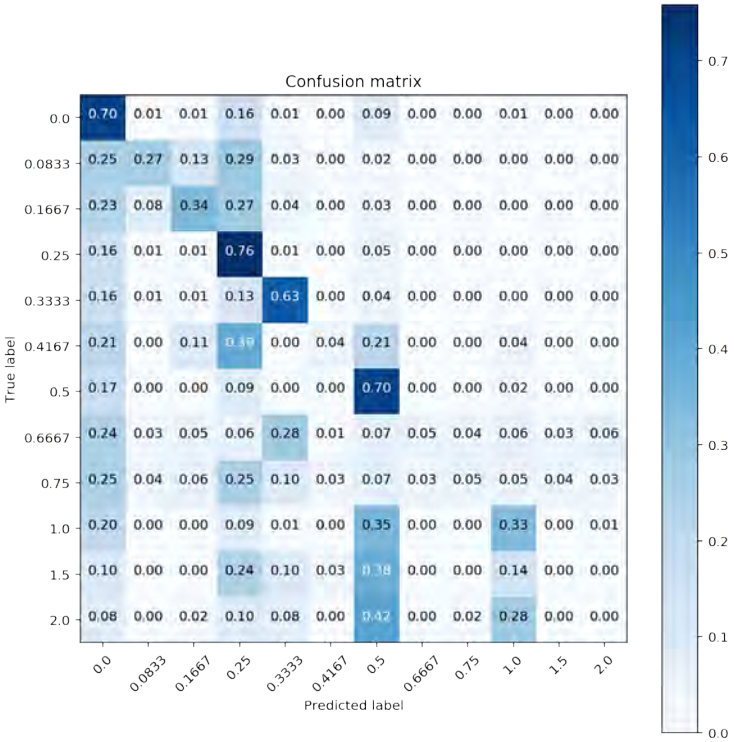

Figure 12: Confusion matrix of the offsetdiffs of LSTM-I

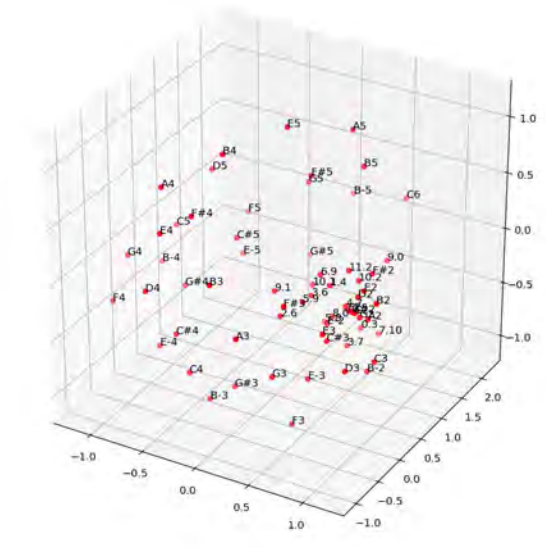

Figure 13: Parameters embedding layer LSTM-I

stopping already stops the process after a little more than 80 epochs.

6.3.1 Performance offsetdiffs

Figure 14 represents the confusion matrix related to the offsetdiffs. We can observe that LSTM-II is more capable of predicting offsetdiffs than LSTM-I, as the diagonal of Figure 14 contains higher values. However, the same problems that were discussed in Section 6.2.1 are still present, only less severe. By comparing the confusion matrix with Figure 8, we see that the model is biased in predicting offsetdiffs with high occurrences, namely: 0.0, 0.25 and 0.5. Moreover, the following values are almost never predicted: 0.4167, 1.5 and 2. These values all occure less than 0.5% of the time.

6.3.2 Performance notes and chords

The weights of the corresponding embedding layer are plotted in Figure 15. The first thing we notice is that chords are prone the be centered on the bottom of the figure, while notes are more spread over the figure. This means both LSTM-I and LSTM-II are capable of distinguishing notes and chords. Furthermore, the octaves seem to increase as we move up the figure. Notes ending with '2' are more present on the bottom, those ending with '3' more in the middle, while the notes ending with '4' and '5' are spread on the top. Where LSTM-I is better capable of recognizing notes of similar frequencies (see Section 6.2), LSTM-II

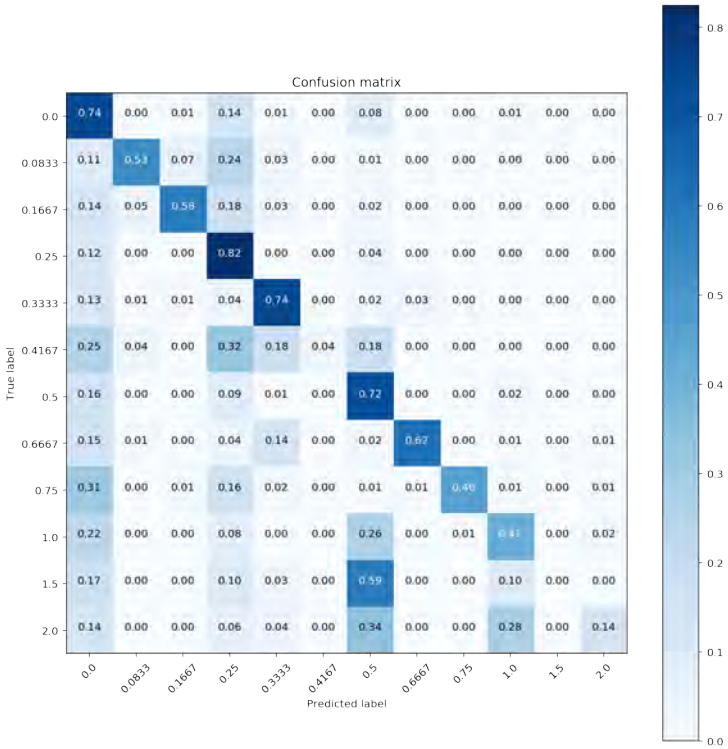

Figure 14: Confusion matrix of the offsetdiffs of LSTM-II

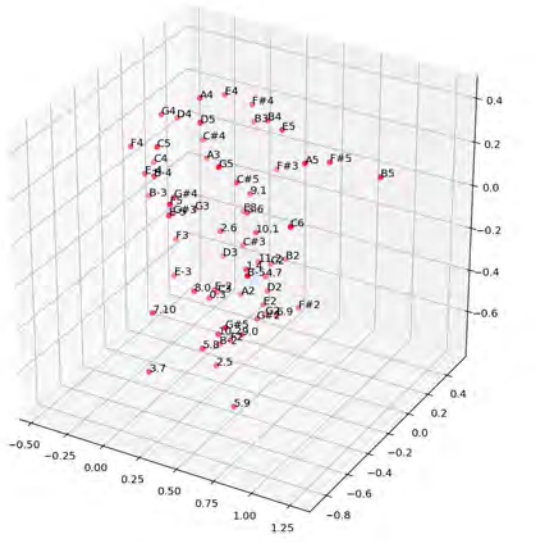

Figure 15: Parameters embedding layer LSTM-II

is better capable of determining a relation between the same notes in different octaves. Examples are: G#3 & G#4, B3 & B4 and C4 & C5.

Thus, LSTM-II better capable of predicting offsetdiffs. However, the patterns concerning notes and chords that are found by LSTM-I are better observable.

6.4 Results CNN

After performing parameter optimization, we select the optimal CNN model instance with: $p_{dr} = 0.5$, $\lambda_{l2} = 0.001$, $lr_0 = 0.005$ and $f = 4$. Figure 11 shows the development of the training and validation losses over time. We can observe that the loss of the training set decreases fast at the beginning and only has a slow decrease after 40 epochs. Thus, the most important patterns are found within the first 40 epochs. After this point, important relationships are still found, as the validation loss is still decreasing. However, these patterns seem to be far less significant.

6.4.1 Performance offsetdiffs

We start by analyzing the confusion matrix of the offsetdiffs, shown in Figure 16. The CNN model seems to perform better than LSTM-I but worse than LSTM-II. The diagonals of the confusion matrix of CNN are generally higher than diagonals of LSTM-I, but lower than the values of LSTM-II. Again, there is a bias towards the values 0.0, 0.25 and 0.5 and a number of values are almost

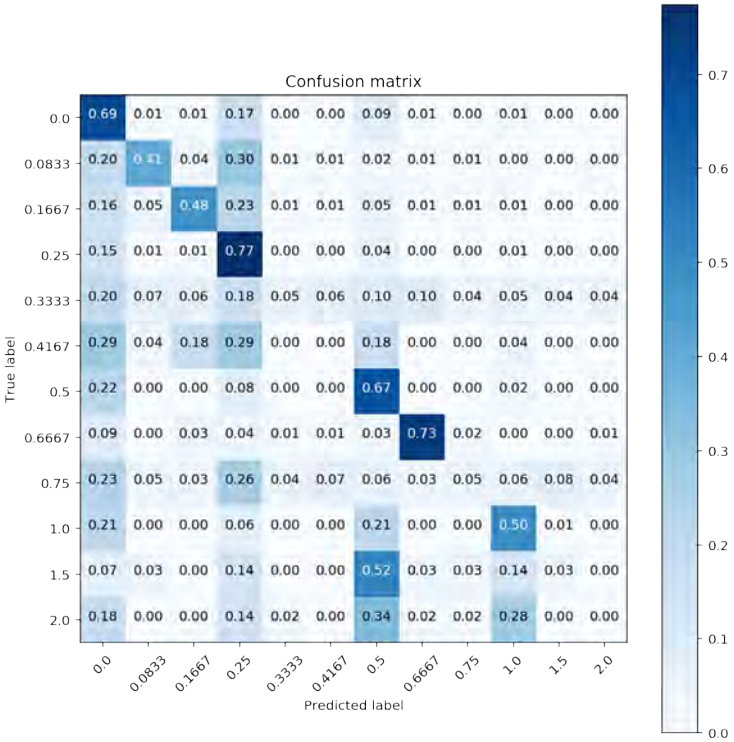

Figure 16: Confusion matrix of the offsetdiffs of CNN

never observed. By looking at Figure 8, we can observe that offsetdiffs with low occurrences are less likely to be predicted. Also, there seems to be more variation in predictions compared to the RNNs. We can conclude this by the fact more cells have probabilities larger than 0.0, compared to the other two confusion matrices (Figure 12 and 14).

6.4.2 Performance notes and chords

Moreover, we will analyze the weights of the embedding layer, (like we did for LSTM-I and LSTM-II in Sections 6.2 and 6.3) in Figure 17. We can observe less of a distinction between chords and notes, as both objects seem to be spread over the figure. Overall, the clusters seem to be less dense. Furthermore, we can see a distinction between the notes in different octaves. Notes ending with '3' are more present on the right top, wile those ending with '5' on the left bottom. Notes ending with '2' and '4' are spread between those two clusters. Finally, we can see that the model occasionally finds a pattern between the same notes in different octaves, for example between F3 and F4 on the bottom.

Although CNN is reasonable capable of predicting offsetdiffs (compared to LSTM-I and LSTM-II), it performs worst in categorizing notes and chords.

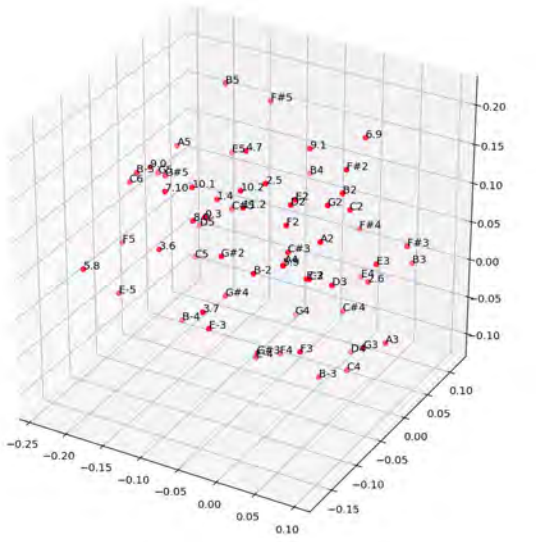

Figure 17: Parameters embedding layer CNN

6.5 Overall results MIDI NNs

Table 1 summarizes the results of the models related to MIDI files. The first thing we notice is that all NNs perform better than the Markov chain. Thus, they are able to recognize more complex patterns, next to the most simple ones. In addition, we can see that LSTM-I and CNN have similar results for the three datasets, meaning they are almost equally capable of recognizing relationships based on historical time steps. LSTM-II on the other hand, has the best performance for the training set, but most importantly for the validation and test set. Lower losses for the validation and test set mean LSTM-II is better at recognizing relationships that are generalizable.

Table 1: Losses per model for different sets

	training	validation	test
LSTM-I	1.84	1.92	1.92
LSTM-II	1.52	1.74	1.75
CNN	1.87	1.96	1.95
Markov chain	2.20	-	2.33

Lastly, LSTM-II seems to perform best in predicting offsetdiffs, followed by CNN and LSTM-I (see Figures 12, 14 and 16). Whereas LSTM-I performs best in categorizing notes and chords, followed by LSTM-II and CNN (see Figures 13, 15 and 17).

6.6 Listening session I

Next, we will analyze the music samples of our MIDI NNs. As stated in Section 5.2, we will conduct two experiments. The first experiment compares samples from the MIDI NNs with samples from the Markov chain. The second one compares audio files from the best performing MIDI NN with files produced by the other two NNs. Table 1 shows that LSTM-II has the lowest test loss and, therefore, analytically performs best. Consequently, the second experiment will compare the samples of LSTM-II with samples of LSTM-I and CNN. After we compared the sets of the two experiments ourselves, we could not make extremely obvious conclusions about difference of quality. We, therefore, proceed by using volunteers to give their judgment. A total of 50 people will attend the listening sessions, who are equally divided over the two experiments.

For both experiments, we will first test if a music set is preferred over the other. There is a preference towards a set if $\mu \neq 0.5$. Where μ represents the population mean of $PS_1^A, PS_2^A, ..., PS_J^A$. In the two following sections, we will test $H_0 : \mu = 0.5$ versus $H_1 : \mu \neq 0.5$, where $\alpha = 0.05$. First, we will see if the t-test is applicable, because parametric tests have more statistical power than non-parametric tests (Campbell & Swinscow (2016)). Statistical power is the likelihood that a experiment will find an effect when there is an effect to be found. However, the test assumes normality in the underlying distribution. To test this assumption, we will analyze the corresponding histogram and quantiles. Furthermore, we will perform Shapiro–Wilk tests (Shapiro & Wilk (1965)), where we will test $H_0 : PS_1^A, PS_2^A, ..., PS_J^A$ *is normally distributed* versus $H_1 : PS_1^A, PS_2^A, ..., PS_J^A$ *is not normally distributed* ($\alpha = 0.05$). If we can not assume normality, we will use the sign test. As stated by Usman (2015), the sign test is a reasonable alternative for the t-test when there is uncertainty about the underlying distribution.

Next, we will examine to what extent the different sets sound like actual music. For every experiment, we will analyze the distribution of the two sets their ACR scores. By calculating the mean, we can see how well the audio files of the sets generally resemble music. Moreover, by looking at the underlying distributions, we can analyze how the opinions of the volunteers are divided.

Lastly, we will analyze the reasons behind the ACR scores. For each of the possible five scores, we will summarize the explanations behind them. This allows us to examine the characteristics of good and bad music. Consequently, we can conclude which patterns are important to find for a model.

6.6.1 MIDI NNs versus Markov chain

In the first experiment, we will analyze the results of the comparison between the MIDI NN files and the Markov chain files. PS_j^A refers to the preference score of volunteer j towards the MIDI NNs. The opinions of the volunteers are plotted in Figure 18. The first thing we notice is that there seems to be a preference towards the MIDI NNs. This is because most preferences scores are larger than 0.5 and the mean equals 0.62. Furthermore, the data seems to

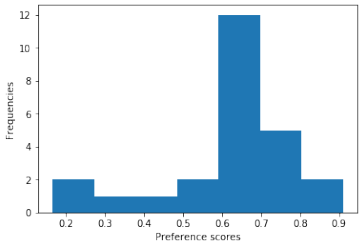

Figure 18: Histogram preference scores first experiment

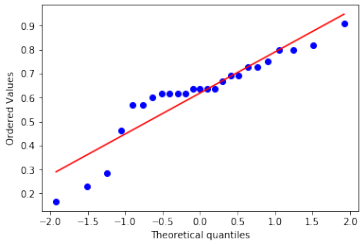

Figure 19: Q-Q plot preference scores first experiment

be skewed to the right and therefore not symmetrical, indicating non-normality. This notion is further strengthened by Figure 19. In this figure, the quantiles of the preference scores are plotted against the quantiles of the normal distribution (also known as the normal Q-Q plot). In case of normality, the quantiles would lie on a straight line, which is not the case. Finally, the resulting p-value of the corresponding Shapiro-Wilk test equals 0.006. Consequently, we can reject the null hypothesis and conclude the preference scores do not follow a normal distribution.

As a result, we will use a sign test to conclude if there is a preference towards a set. The resulting p-value is 0.0. We, therefore, can reject the null hypotheses. Thus, there is a preference towards one of the two sets. Because the mean preference score is 0.62, we can conclude the MIDI NNs provide more music-resembling audio. Hence, NNs can find patterns in music that our baseline can not, and these patterns can by noticed by the human ear. Furthermore, the Markov chain having the highest loss values (see Table 1) indicates a negative relationship between the loss values and produced music quality.

Figures 20 and 21 summarize the resulting ACR scores. If we round the mean ACR scores and translate the values (see Section 5.2), we derive the absolute performances. The mean rounded ACR scores are 3 and 2 for the MIDI NNs and the Markov chain, respectively. Thus, the MIDI NNs perform fairly and the Markov model poorly. Lastly, we can see that there is more variations in the ACR scores of the Markov model compared to the MIDI NNs. This indicates that the opinions about the audio files of the Markov model are more divided.

6.6.2 LSTM-II versus LSTM-I & CNN

The second experiment relates to the comparison between the best performing MIDI NN (LSTM-II) and the worse performing MIDI NNs (LSTM-I & CNN). In this section, PS_j^A will relate to the preference score of volunteer j towards LSTM-II. First, we will examine if we can assume that the preference scores are normally distributed. The histogram of $PS_1^A, ..., PS_J^A$ is given in Figure 22. There is a clear peak around $x = 0.4$ and the mean equals 0.42. Although there seems to be approximately as many points on each side of this peak, it is hard

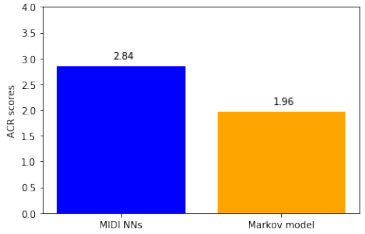

Figure 20: Average ACR scores first experiment

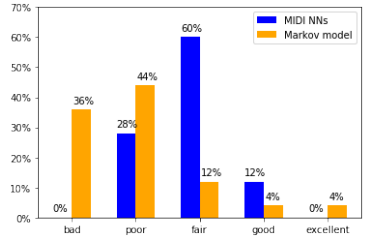

Figure 21: Distribution ACR scores first experiment

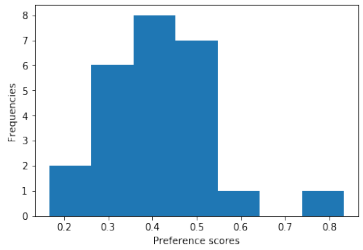

Figure 22: Histogram preference scores second experiment

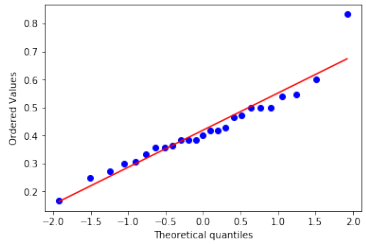

Figure 23: Q-Q plot preference scores second experiment

to conclude symmetry. This is because the points are distributed differently on each side of the peak. The normal Q-Q plot (shown in Figure 23) provides us more insight. The quantiles seem to lie on a straight line, except for the most top-right point, which we will view as outlier. This indicates that the preference scores are normally distributed. Lastly, the resulting p-value of the Shapiro-Wilk test equals 0.12. This means we can not reject the null hypotheses and conclude the data is not normally distributed. Thus, it is reasonable to assume normality.

Hence, we will use a t-test to conclude if there is a preference towards a musical set. The resulting p-value equals 0.006. This means we can reject the null hypotheses and conclude there is a preference towards one of the two sets. Because the average preference score equals 0.42, we can say that LSTM-II produces less music-resembling audio than LSTM-I and CNN. This contradicts the notion of models with lower losses producing better music.

Finally, we will analyze the ACR scores of the second experiment. Figure 24 shows the average evaluations of the music sets. We can see that LSTM-II is again being outperformed by LSTM-I and CNN. This further indicates LSTM-II producing less qualitative music samples. Moreover, LSTM-II's music samples are generally between poor and fair and LSTM-II and CNN their samples are generally received as fair. The distributions of the related ACR scores are given in Figure 25. We can see that there is not much variation in the ACR scores of LSTM-II and most people either give its samples a poor or fair rating. If we

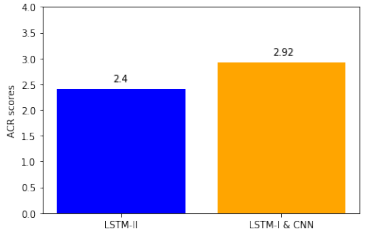

Figure 24: Average ACR scores second experiment

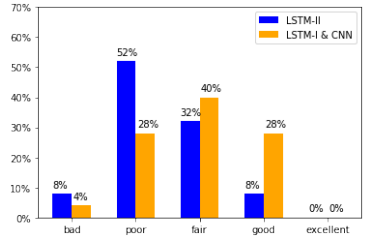

Figure 25: Distribution ACR scores second experiment

look at the ratings of LSTM-I and CNN their audio files, we can see that there is more spread in the underlying distribution. This indicates that these opinions are more divided than the opinions about LSTM-II's samples. Furthermore, the distribution of LSTM-I CNN their samples seems to be symmetrical around "fair". This means that there are approximately as many people who give a rating above as below "fair".

6.6.3 Reasons behind ACR scores

The reasons behind the ACR scores of the music sets are divided in two groups. The ones belonging to a score of 1 or 2 are placed in the *"bad evaluated group"*. The corresponding opinions are summarized in Figure 26. We can see that a musical piece that sounds atonal or contains many dissonant notes is the main reason for bad evaluations. This indicates that it is more important to predict the notes and chords correctly than the offsetdiffs. This is because the former has influence on atonality. Other important reasons for bad evaluations are the fact a listener could not recognize a melody nor rhythm and random sounding musical pieces. Furthermore, even if music sets have simple melody or rhythm, they can still have bad ratings. This means that music needs to have more complexity for it to be perceived better.

The other group is the *"fair/good evaluated group"*. These explanations correspond to the ACR scores of 3, 4 and 5 and are summarized in Figure 27. The most important characteristic of the *"fair/good evaluated group"* is variation in notes and tempo. More diversity in a dataset could make it possible for an NN to learn more variation, although it also makes it harder to learn generalizable patterns. Furthermore, the second important charateristic is a recognisable melody, followed by a good rhythm. The former is generally influenced by the notes and chords and the later by the offsetdiffs. This further strengthens the notion that it is more important to predict the notes and chords correctly than offsetsdiffs.

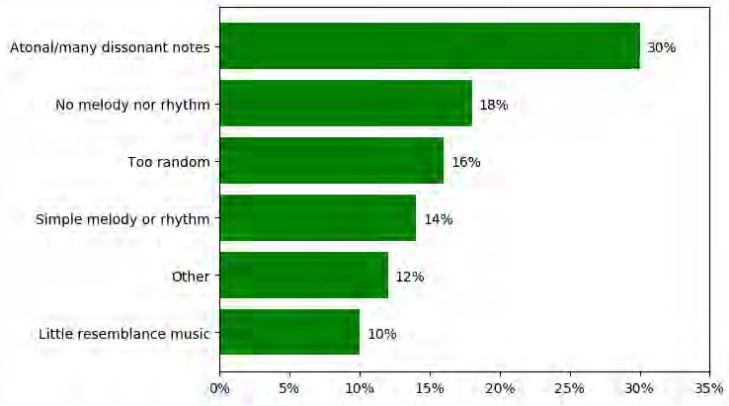

Figure 26: Reasons behind bad evaluated ACR scores

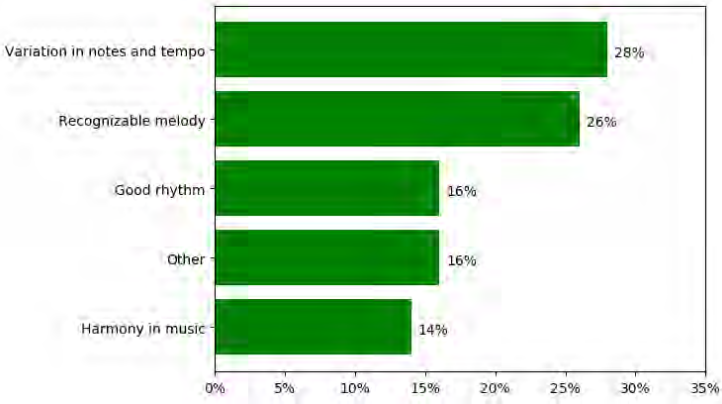

Figure 27: Reasons behind fair/good evaluated ACR scores

Part II
Raw audio

7 Data

In this section, we discuss the necessary steps we will take in order to prepare the techno and classical datasets consisting of WAV files to a usable format. We will start by discussing the structure and details of the datasets. Next, we will provide an explanation on how we convert the audio to numbers. The models we will implement in Section 8 tend to be more complex than the models we implemented in Section 4. We, therefore, will use open-source implementations in order to implement two models: Wavenet[3] (see Section 8.1) and SampleRNN[4] (see Section 8.2). The process of data cleaning will, therefore, partly depend on the implementations and can differ per model. The first steps, however, will be similar and are summarized in Figure 28 (songs relate to the techno music dataset).

Dutch DJ Joris Voorn provided us with a techno music set. Not all songs are suited to be analyzed. A number of songs contain human vocals. For this research we want to solely focus on producing beats first, because we do not know to what extent human vocals have influence on the learning capabilities. Hence, the songs having vocals are removed. Moreover, a number of songs are remixes or variations of an original song already present in the Voorn dataset. These songs can lead to an unrepresentative dataset, because certain sounds are represented too often. Consequently, remixes are deleted. The final set contains 32 songs, equivalent to approximately two hours of music. Furthermore, we will analyze a work of Bach called "The Goldberg Variations"[5], which consists of 32 files with a total length of 1 hour and 22 minutes. They are all performed on a harpsichord. Because, only a single instrument is used, we expect to have less variations of sound to analyze. Hence, we expect our models to have more trouble learning patterns in the techno dataset (provided by Voorn) compared to the classical dataset (composed by Bach). The music is delivered in WAV files, i.e., audio in uncompressed format. All information will be present, unlike mp3 files, for example, where parts are removed to save space. The resulting sound might be indistinguishable from WAV files for a human, but for analytical purposes, WAV files are preferred. By making sure we have as much information as possible, we can analyze the audio files extensively, helping us to ultimately generate music of better quality.

After receiving the music samples, the data must be converted to numbers so that we can start analyzing the audio. To achieve this, we will sample amplitude values at discrete time points given a sample rate s_r of 16,000 (like Mehri et

[3] https://github.com/basveeling/wavenet

[4] https://github.com/deepsound-project/samplernn-pytorch

[5] The files are from "http://freemusicarchive.org/music/Kimiko_Ishizaka/The_Open_Goldberg_Variations". They were in MP3 format and, consequently, converted to WAV

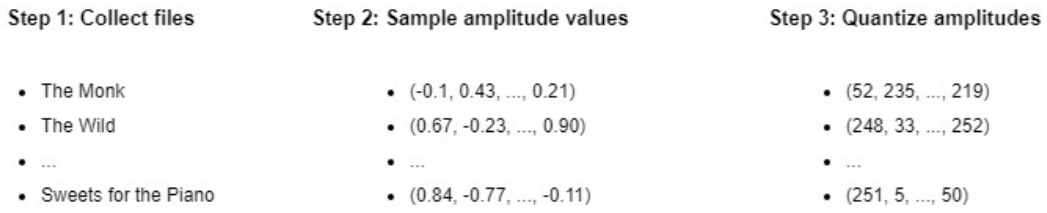

Figure 28: General data preparation steps for WAV files

al.(2016) did). Hence, each audio file will be converted to an array of amplitude values.

Although, amplitude values are stored as continuous values, we would like to convert these values to categories for our NNs. However, raw audio is generally stored using 16-bits. This leads to 65,536 possible values per time step. To make the output easier to grasp, we will convert the data to 256 possible values. This is the exact same number of values van den Oord et al.(2016a) and Mehri et al.(2016) used. We will use two methods to transform the data, namely: μ-law quantization and linear quantization. Equation 20 summarizes the former process, where $-1 < x_t < 1$, $\mu = 255$ and $0 \leq f(x_t) \leq 255$. Equation 21 summarizes the latter process, with $0 < x_t < 1$, $\triangle = 256$, $0 \leq f(x_t) \leq 255$ and ϵ being an arbitrary small value. For Wavenet, we will implement μ-law quantization, while we will implement linear quantization for SampleRNN. The choices of quantization method is based on decisions and results of van den Oord et al.(2016a) and Mehri et al.(2016).

$$f(x_t) = sign(x_t)\frac{ln(1+\mu|x_t|)}{ln(1+\mu)} \tag{20}$$

$$f(x_t) = \lfloor x_t \cdot (\triangle - \epsilon) + \frac{\epsilon}{2} \rfloor \tag{21}$$

In the next two sections, we will describe how we prepare the data for the models separately. This is process depends on the implementations found online and the requirements of the individual models.

7.1 Data preparation Wavenet

First, we will discuss how we prepare the data for Wavenet. Figure 29 summarizes the process. The values at different time steps are categories represented as integers. To prevent our NNs to assume ordering between these integers while still remaining a numerical representation, we will apply one-hot encoding. Each integer will be replaced by a list of binary variables of size 256. Exactly one value in the lists equals "1". The index of this value represents the quantized value of the amplitude (indices starting with 0).

Note that different songs are represented as arrays of values. To feed data to models and evaluate the results, we need to convert these values to time

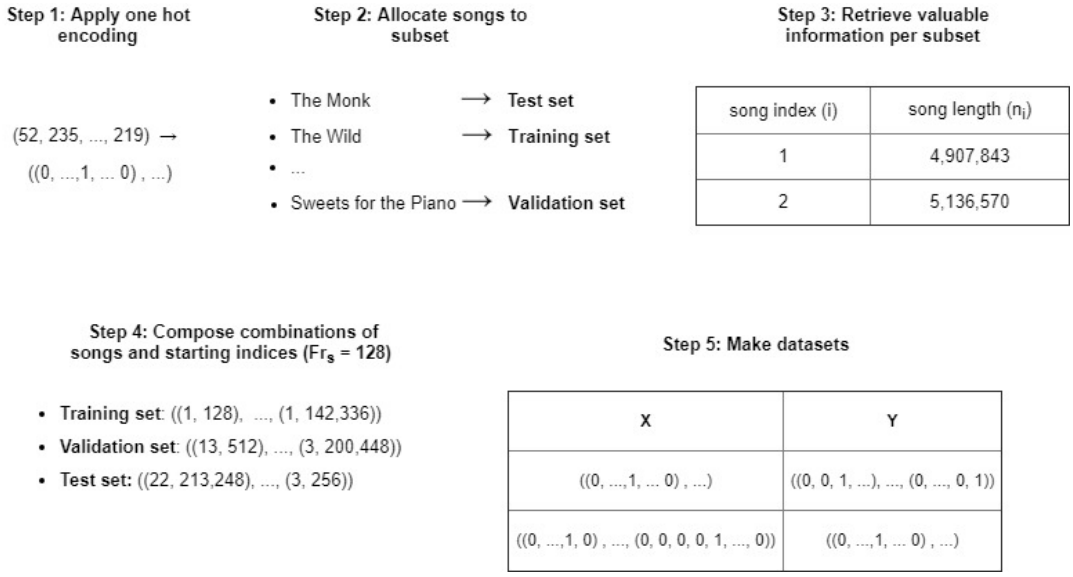

Figure 29: Data preparation Wavenet

steps $\mathbf{X}$ and target values $\mathbf{Y}$. First, we assign a song to a validation, train or test set. It is complicated to assign a specific percentage to a subset. There are only a select number of songs per music genre dataset and the songs have varying lengths. Instead, we will try to come close. For both datasets, we will assign songs to the training, test and validation set with a probability of 64%, 20% and 16%, respectively. The resulting techno subsets contain 61%, 25% and 14% of the data, respectively. The percentages related to the classical files are: 64%, 18%, 18%, respectively. The rows within these datasets should be random, i.e., not depend on the songs or the order within songs. For each of the genre-specific subset, we first give every song an integer i. The array length of song i is denoted as n_i. Next, we define a fragment stride fr_s. In the following step, we loop over all songs and make combinations between song i and starting indices q_p^i ($p = 0, 1, .., (P-1)$ and $P = \lfloor \frac{n_i}{fr_s} \rfloor$). If we would define $fr_s = 128$ and apply the procedure to an array of song i with $n_i = 10,000$, the resulting values would be: (i, 0), (i, 128), (i, 256), ..., (i, 9,856). The resulting values for all songs are appended to the same list and shuffled, eliminating a logical ordering. Similarly to the datasets we constructed in Section 3, $\mathbf{X}$ will be represented by a sequence of S succeeding time steps. The number of features per time step will equal 256 (the length of the one-hot encoded vector). Every combination of i and q_p^i will be used to construct a row in the related subset. For every combination, we use array i to select time points q_p^i, q_{p+1}^i, ..., q_{p+S}^i as a row and q_{p+1}^i, q_{p+2}^i, ..., q_{p+S+1}^i as the corresponding target values. Note that $\mathbf{X}$ and $\mathbf{Y}$ both contain S columns.

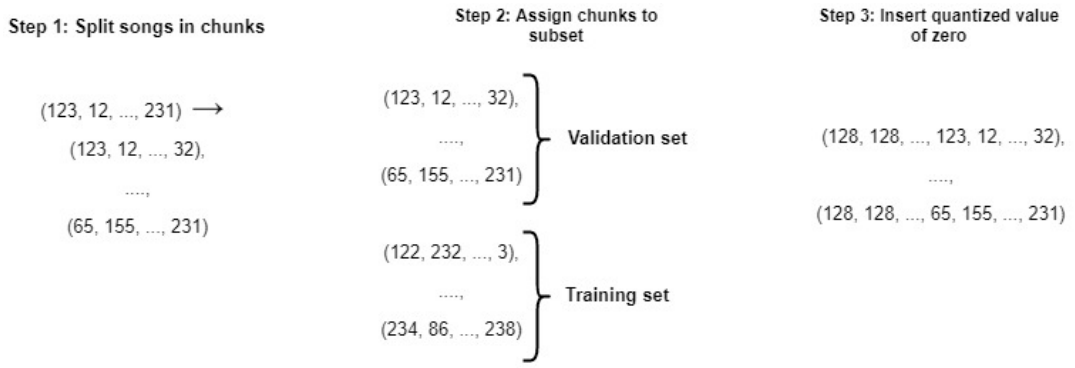

Figure 30: Data preparation SampleRNN

7.2 Data preparation SampleRNN

The overall process of preparing the data for SampleRNN is shown in Figure 30. Unlike the preparation for Wavenet, we will only apply one-hot encoding to the target values $\mathbf{Y}$. For the time steps $\mathbf{X}$ on the other hand, we will use an embedding layer (see Section 4.1.2) to handle the categories adequately (see Section 8.2.1).

Next, every song is split up in arrays of length 128,000, equivalent to $\frac{128.000}{s_r} =$ 8 seconds. The chunks will be assigned to a training, test or validation set. We have a total of 609 classical chunks. Ideally, we would like to have approximately the same number of techno chunks. We, therefore, only use a subset of the techno dataset to produce chunks. The total of resulting files equals 623. Chunks belonging to a single song will be present in the same genre-specific subset, whereas songs will be randomly assigned to a dataset. After the chunks are saved, they are randomly shuffled. Ideally, we would like to assign 64%, 20% and 16% to the training, test and validation set, respectively. Again, a specific percentage per subset is hard to reach for us. Songs are assigned to relevant subsets based on preferred division. For the techno files, the resulting percentages are: 60 %, 25% and 15% for the training, test and validation set, respectively. The resulting percentages of the classical files are 63%, 19% and 18%, respectively.

The following steps consider the construction of $\mathbf{X}$ and $\mathbf{Y}$. SampleRNN uses K modules that all consider their own set of input values (see Section 8.2). Each module operates on non-overlapping frames of size FS^k $(1 \leq k \leq K)$. In order for the model to calculate each value in $\mathbf{Y}$, we add $\prod_{i=2}^{K} FS^i$ values of O to the left side of each chunk. We give O the value of the quantized value of 0, or in other words: $O = \frac{\triangle}{2} = 128$. The determination of $\mathbf{X}$ and $\mathbf{Y}$ will be further discussed in Section 8.2.2, after we explain the set-up of the model.

Figure 31: Causal convolutional layers

8 Methods

Next, we will discuss the methods we will implement to analyze WAV files. The related models will analyze raw audio wave forms, instead of notes, chords and offsetdiffs. Consequently, they will be more complex and computationally expensive.

8.1 Wavenet

The first model we will use to analyze WAV files is Wavenet. The model was already briefly discussed in Section 2.1. The main idea of Wavenet is to predict an audio sample x_t based on preceding samples by using a network of CNNs. x_t is, therefore, conditioned on the samples at previous points in time, as shown in Equation 1. The network contains no pooling layers and the input and output have the same time dimensionality throughout.

The subsequent value x_t will be estimated by using a softmax layer, as already discussed in Section 4.1.3. Using a softmax distribution on raw audio waves might not seem intuitive, but van den Oord et al. (2016a) already showed that they perform adequately even when the data is implicitly continuous.

The main power of Wavenet lies in the causal convolutions. The causal convolutions make sure the model can not depend on future points in time and, thus, ensures that time ordering is preserved. Or in other words, we can only predict x_t using preceding time points $x_1, x_2, ..., x_{t-1}$. Figure 31 shows several causal convolutional layers. In this example, the filter length equals 2, the stride equals 1 and no padding is used. The receptive field of models using causal convolutions layer depends on the depth. The higher the depth, the higher the receptive field.

However, the problem with causal convolutions is that they require many layers or large filters in order to have a receptive field that is reasonably large. To solve this problem, van den Oord et al.(2016a) introduce dilated causal convolutions. In dilated causal convolutions, the filter skips input values by certain steps, resulting in larger receptive fields, without using to many layers. The idea is summarized in Figure 32, where the dilations equal 1, 2 and 4 for the first, second and third layer, respectively. The dilation depth D equals 2 in this example. Exponentially increasing the dilation factor leads to an exponential

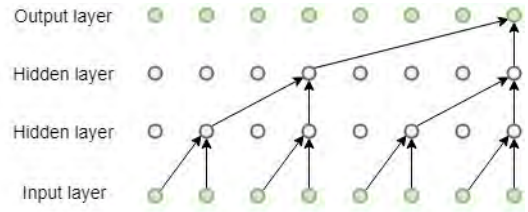

Figure 32: Dilated causal convolutions

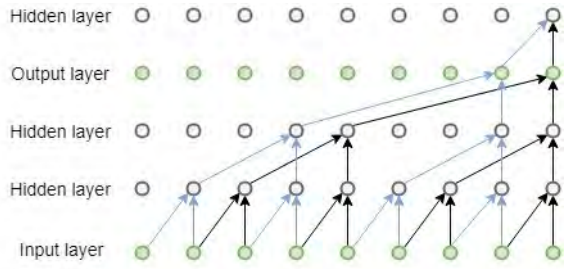

Figure 33: Wavenet stacks

increase of the receptive field [29], and thus the capacity of the model.

Furthermore, a Wavenet model may contain a number of stacks, denoted as nb_s. After each stack of layers, the dilation rate will be set to one. A specific stack can also be viewed as an individual Wavenet sub-model. A Wavenet model with $nb_s = 3$ and $D = 2$ would, for example, have the following dilations: 1, 2, 4, 1, 2, 4, 1, 2, 4. Figure 33 shows the first five layers of this network. The bottom four correspond to the first stack and the fifth layer to the second stack.

Moreover, Wavenet uses a gated activation unit, that was first introduced by van den Oord et al. (2016b). In their corresponding paper, the authors make a comparison of their CNN based model PixelCNN versus the LSTM counterpart PixelRNN. They argue that PixelRNN has two main advantages. Firstly, the layers in the LSTM models have access to the entire neighbourhood of points. Whereas, the access is limited for PixelCNN. However, this disadvantage can be alleviated by using many layers. Secondly, PixelRNN can benefit from the multiplicative units to learn more intricate patterns, because of PixelRNN its use of LSTMs. In their effort to overcome these shortcomings of CNNs, van den Oord et al. (2016b) replace the rectified linear units with a gated activation function, given in Equation 22. Here $*$ denotes the convolutional operator, $\odot$ an element-wise multiplication operator and W a learnable convolution filter. The filter f and the gate g correspond to the set of weights of the two activation gates: sigmoid (σ) and tanh, respectively.

$$\mathbf{z} = tanh(W_{f,k} * \mathbf{x}) \odot \sigma(W_{g,k} * \mathbf{x}) \tag{22}$$

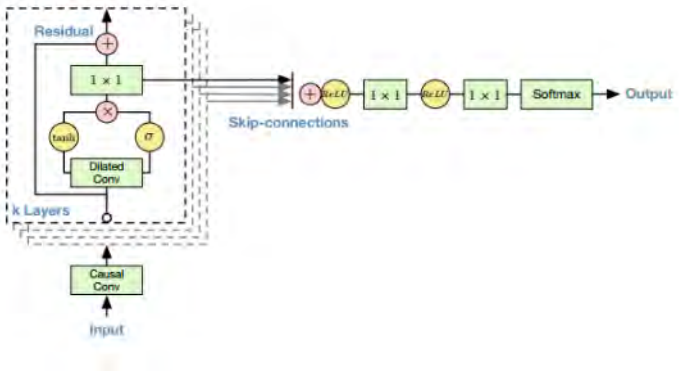

Figure 34: Architecture Wavenet

Several papers have already explored the use of gated activation function in NNs, for example: Kalchbrenner et al. (2015), Kaiser & Sutskever (2015) and Srivastava et al. (2015). This type of activation function generally performs better.

8.1.1 Architecture Wavenet

Also, we will highlight the overall architecture of the Wavenet model in depth. Before the input is passed to the hidden layers, an initial causal convolution is applied. In every hidden layer two methods, that were first introduced by He et al. (2018), are used, namely: residual mapping and skip connections. The main purpose of these techniques is making deeper networks easier to optimize. In essence, they increase the gradient flow within the network by extending the capacity of a model. And most importantly, this is all done without increasing the number of parameters too much.

In every layer, $\mathbf{x}$ passes through Wavenet's activation function (Equation 22) and a 1-D convolution of kernel size 1 (also known as 1×1 convolution) to produce $F(\mathbf{x})$. The original value $\mathbf{x}$ is added to the resulting output $F(\mathbf{x})$. The summation is thereafter passed on to the next convolutional layer. Lastly, $F(\mathbf{x})$ is added to a list of outputs by skip-connections.

After the last hidden layer, the list of $F(\mathbf{x})$ values is added to the current output values. Next, two succeeding 1×1 convolutions and a softmax layer are followed. Finally, the softmax layer will return S arrays of probabilities of length 256. The whole architecture is summarized in Figure 34.

8.1.2 Input and output

Before we start training our Wavenet model, we have to determine the sequence length S. We determine S by using equation 23.

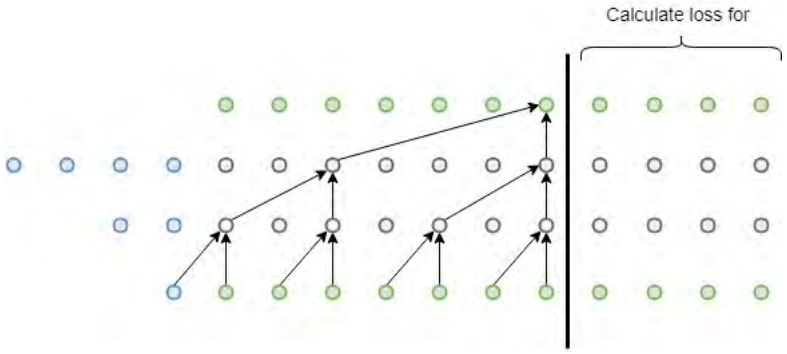

Figure 35: Zero padding in Wavenet

$$S = 8192 + nb_s \cdot 2^{D+1} - (n_s - 1) \tag{23}$$

Van den Oord et al. (2016c) state that the input and output should have the same length for all layers, namely S. To ensure this requirement, we will use zero padding. For every layer, we add Q time steps on the left side of the current input. The added time steps are represented as arrays of zeros. In addition, Q equals the dilation rate of the layer. Thus, the first layer will add 1 time step, the second 2 time steps, the third 4 time steps and so on.

Note that zero padding is at least applied in one layer for the first $nb_s \cdot 2^{D+1} - (n_s - 1) - 1$ nodes. These corresponding outputs will, therefore, provide less insights, as less information is used to calculate them. Consequently, we will only consider the last $8192 + 1 = 8193$ values of the output to calculate the loss, although the output will have a length of S.

Figure 35 summarizes the process of padding and selected output nodes. The padding nodes are given in blue. In this example, $D = 2$, $nb_s = 1$ and $S = 11$. The number of input nodes where padding is applied, is calculated as following: $nb_s \cdot 2^{D+1} - (n_s - 1) - 1 = 2^3 - 0 - 1 = 7$. This means the loss function considers $11 - 7 = 4$ values for every input.

8.1.3 Parameter settings

Due to the computational expensiveness of Wavenet models, we are not capable of fully examining parameter settings. Instead, we determine most of the parameters based on the restriction of our computational capabilities. Other insights come from Wavenet implementations for music generation found online [6,7] and our own insights. To train our Wavenet models, we will use an Adam optimizer where $lr = 0.001$. The errors will be calculated using categorical cross entropy (see Equation 10). The fragment stride fr_s controls the number of rows

[6] https://github.com/basveeling/wavenet
[7] https://github.com/ibab/tensorflow-wavenet

in the datasets. To make sure the classical training set and the techno training set have approximately the same amount of data, we set $fr_s = 8192$ for the techno sets and $fr_s = 4096$ for the classical sets. This eventually results in 12736 rows in the techno training set and 11760 rows in the classical training set. The number of filters in each convolutional layer will equal 64, except for the last two 1×1 layers. The first of the two layers summarizes the information received from the skip-connections and uses 1024 filters. The output dimensions of the second layer must equal the output dimension of the softmax layer, and is therefore set to 256. The dilation depth D will equal 11 and the number of stacks nb_s 4, leading to a receptive field of 1023.8 milliseconds. Also, we will reduce overfitting by implementing early stopping (see Section 4.1.4), where the patience equals 10. Finally, the models will run for 100 epochs (unless early stopping terminates the training procedure).

8.2 SampleRNN

The final model we use in order to generate music is SampleRNN. Section 2.2 already provided some background about this model. Like Wavenet, SampleRNN predicts raw audio waveforms based on historical values. The value of a time point x_t is based on conditional probabilities of preceding time points, as shown in Equation 1. In contrary to Wavenet, SampleRNN primarily uses RNNs instead of CNNs. Mehri et al. (2016) state that the challenge in modeling waveforms lies in the fact they generally contain correlation at different time scales. SampleRNN's strength lies in the fact it can capture patterns between sequences that are far away from each other as well as ones that are close to each other. The architecture of SampleRNN enables this capability by consisting of a hierarchy of modules (also called tiers). Each of them operates at a different temporal resolution, i.e., each module considers a different number of succeeding time points. The architecture of SampleRNN is summarized in Figure 36.

The higher a module in the hierarchy, the lower its temporal resolution. The RNNs in the higher modules consider more time points as input and try to find relationships between different sets of these inputs. Thus, modules higher in the hierarchy search for patterns between time points further away. Each module passes its output to the one below it, where the module on the bottom of the architecture predicts on a time-point level. The modules above the lowest layer, however, operate on non-overlapping frames f_k of size FS^k, where k represents the level of the module in the hierarchy. The bottom tier corresponds to $k = 1$ and the top tier to $k = K$. Because the temporal solution decreases when we move up the architecture, an RNN in higher tiers requires fewer time units to process all time points. Thus, a value of t has a different meaning per module. Whenever we refer to t in an equation, we refer to the value related to tier k in that same in equation. In Figure 36, for example, a value of $t = 1$ refers to the input $x_1, ..., x_4$ for tier 2 ($FS^2 = 4$). Whereas, $t = 1$ corresponds to $x_1, ..., x_{16}$ for tier 3, because $FS^2 \cdot FS^3 = 4 \cdot 4 = 16$. The ratio between the temporal solutions of tier k and $k - 1$ is denoted as r^k. Notice that r^3 and r^2 both equal 4 in Figure 36.

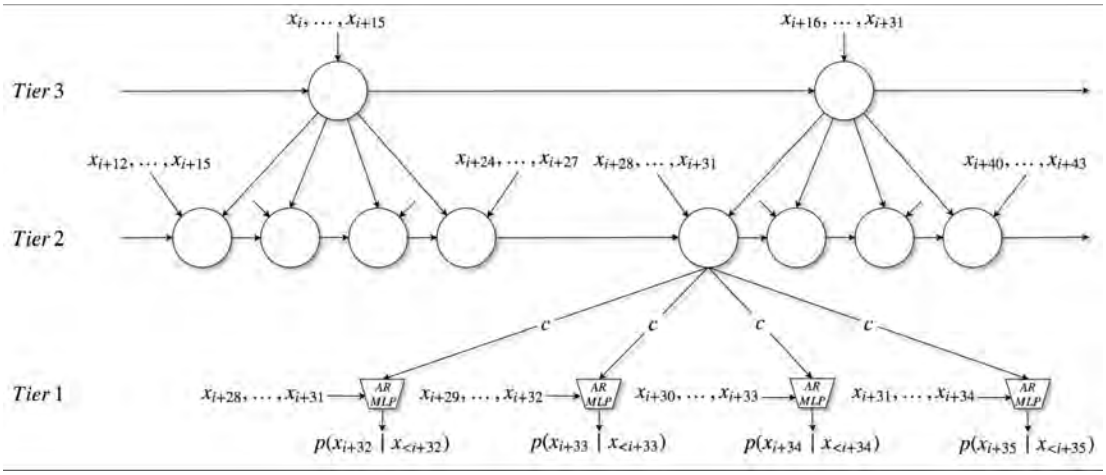

Figure 36: Architecture SampleRNN

A tier can have multiple RNNs (for $k \geq 2$). The RNNs output a hidden state h_t^k based on h_{t-1}^k and an input inp_t^k (see Equation 24). The input at the top module inp_t^K solely depends on the input frame f_t^K. Whereas the input for the intermediate modules ($1 < k < K$) depends on the corresponding input frame f_t^k and the information it receives from the tier above it c_t^{k+1} (see Equation 25). Furthermore, the modules operating on different time scales means a module first has to upsample the output before it can pass it downwards. Or in other words, a module k ($k \geq 2$) needs to match the time scale of the module below it. The relating procedure is summarized in Equation 26.

$$h_t = \mathbf{H}(h_{t-1}, inp_t) \tag{24}$$

$$inp_t = \begin{cases} W_x f_t^k + c_t^{k+1} & 1 < k < K \\ f_t^K & k = K \end{cases} \tag{25}$$

$$c_{(t-1)*r+j}^k = W_j h_t, \quad 1 \leq k < K \tag{26}$$

8.2.1 Tier 1

The module on the bottom of the architecture (k = 1) predicts the value for the next time point x_{t+1}. The estimated probabilities are based on the FS^1 preceding samples and the vector c_t^2. Mehri et al. (2016) argue that the sequence $x_{t-FS^1+1}, ..., x_t$ is easy to model by a simple memoryless model, because FS^1 is often small and correlations between nearby time points are easier to find. They, therefore, implement a multilayer perceptron (MLP) in the first tier. Unlike the subsequent tiers, which all contain RNNs. The architecture of the first tier is given in Figure 37.

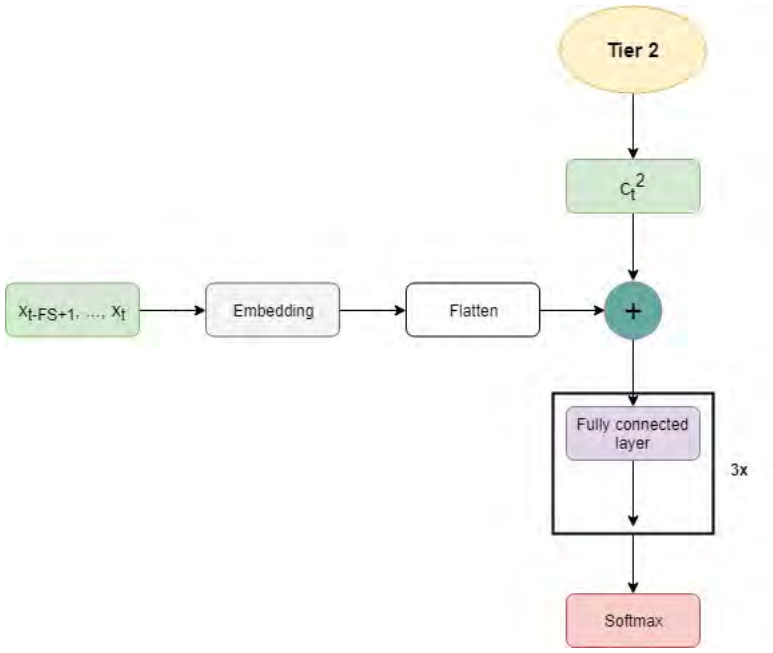

Figure 37: Architecture tier 1 SampleRNN

First, the previous FS^1 time points pass through an embedding layer (see Section 4.1.2) and a flatten layer to produce a 1-dimensional vector. After which the output of tier 2 is added by Equation 25. The output passes through three fully connected layer. The last of these three layers contains $\triangle = 256$ neurons to prepare the data for a final softmax layer ($T = 1$). Finally, the probabilities $\{\hat{y}^c_{t_1+1} : 1 \leq c \leq \triangle\}$ are calculated.

8.2.2 Input & output

Mehri et al. (2016) concluded that using 3 tiers is generally optimal. We, therefore, will use three tiers as well. Each chunk contains $\prod_{i=2}^{3} FS^i + 8 \cdot s_r = \prod_{i=2}^{3} FS^i + 128,000$ time points (see Section 7.2). For each chunk, we consider a value x_{t+1} as target value if $t + 1 > \prod_{i=2}^{3} FS^i$. Tier 1 takes sequence $\{x_{t-(FS^1-1)}, ..., x_t\}$ as corresponding input, while f_t^2 equals $\{x_a, x_{a+1}, ..., x_{a+(FS^2-1)}\}$ and $a = (\lfloor \frac{t+1}{r^2} \rfloor - 1) \cdot r^2$. Lastly, tier 3 considers $\{x_b, x_{b+1}, ..., x_{b+(\prod_{i=2}^{3} FS^i-1)}\}$ as input, where $b = (\lfloor \frac{a+r^2}{r^2 r^3} \rfloor - 1) \cdot r^2 r^3$

8.2.3 GRU models

The type of RNNs we will use for SampleRNN are GRUs and were first introduced by Cho et al. (2014). GRUs have similarities to LSTMs (see Section 4.1). Both models use a unit with different gates and are known to limit the effect

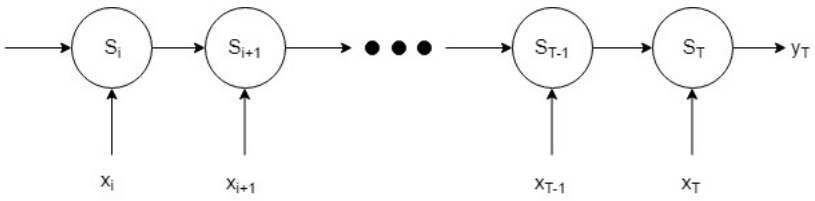

Figure 38: Unrolled RNN

of the vanishing gradient problem. Moreover, GRUs are better suited to store long-term information compared to standard RNNs.

The memory cell contains gates, namely: an update gate and a reset gate. Intuitively, the update gate determines to what extent information should be passed on and the reset gate to what extent information should be forgotten. Their corresponding formulas are given in Equations 27 and 28, respectively.

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{27}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + bz) \tag{28}$$

where x_t describes the input at time point t, h_t the output vector of the GRU unit and b_q the bias. In addition, W_q and U_q correspond to the weights of the input and recurrent connections, respectively.

Next, the new memory content $h_t^{'}$ is calculated using Equation 29. $h_t^{'}$ uses r_t to store relevant information from the past.

$$h_t^{'} = tanh(W x_t + r_t \odot h_{t-1}) \tag{29}$$

Finally, the output vector h_t is determined by using the update gate (see Equation 30).

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h_t^{'} \tag{30}$$

Although LSTM and GRU have similar advantages and structure, Mehri et al. (2016) state that SampleRNN performs slightly better when it uses GRUs.

8.2.4 Truncated Backpropagation Through Time

Backpropagation Through Time (BTT) is the algorithm that is generally used to train RNNs. During training, an RNN is unrolled, where each time step can be viewed as an additional layer within the network (see Figure 38). The inputs are passed through the network to calculate a final output. An error is computed and the weights are updated according to this error.

Mehri et al. (2016) state that training RNNs on long sequences can be a computationally expensive task. Furthermore, RNNs can suffer from vanishing gradients. In spite of this disadvantage, RNNs have proved to be powerful

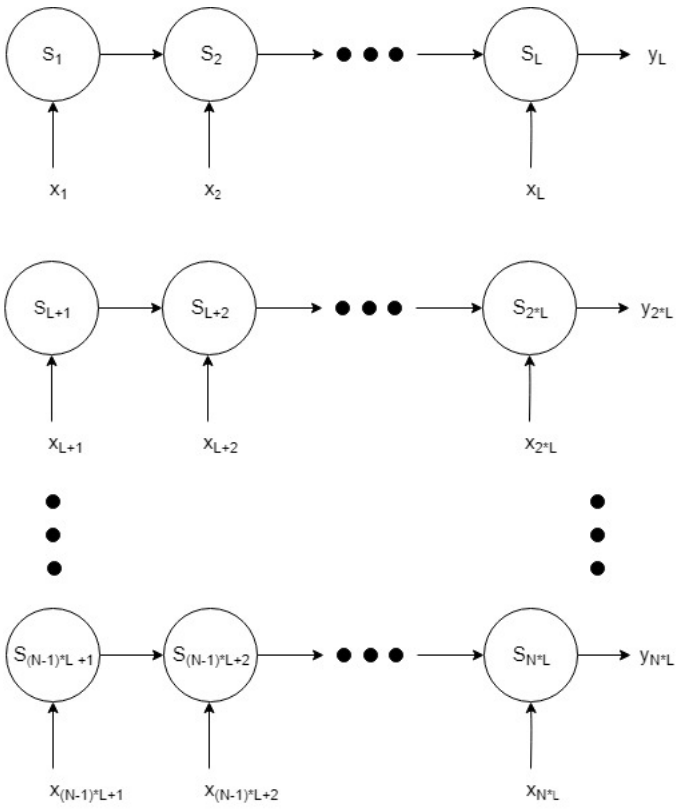

Figure 39: Unrolled subsequences TBTT

models. Mehri et al. (2016) implement truncated backpropagation through time (TBPTT) to speed up the training and limit the gradient of vanishing. During this procedure, every sequence is split in N subsequences of length L. Each subsequence calculates an output and the corresponding error, as shown in Figure 39, and backpropagation is only applied to a subsequence.

8.2.5 Parameters set

Because of our lack of computational resources, we will not perform any parameter optimization for the SampleRNN model. Instead, we base our choices on the authors (Mehri et al. (2016)) and others[8] (who implemented SampleRNN) their findings. The models will be trained using an Adam optimizer, using a constant learning rate lr of 0.001. The errors are calculated using categorical cross entropy (See Equation 10). Tier 2 and 3 will contain 2 GRUs. Additionally, $r^2 = 16$ and $r^3 = 4$. Furthermore, $FS^2 = FS^1 = 16$ and $FS^3 = 4$. The

[8]https://github.com/deepsound-project/samplernn-pytorch

output dimension of the embedding layer e_l in tier 1 will equal 256. All neural networks in the architectures (the RNNs and the fully connected layers) contain 1024 neurons. Except for the last fully connected layer that prepares the current output for the softmax layer and, therefore, uses 256 neurons. Lastly, the length of the subsequences will be set to $L = 1024$ time points during TBTT. This means tier 1, 2, and 3 use $L = 1024$, $\frac{L}{FS^2} = 64$ and $\frac{L}{FS^2 \cdot FS^3} = 16$ frames for each round, respectively. Lastly, the SampleRNN models instances will run for 20 epochs.

9 Experiments

Next to the MIDI file models, we will would like to examine the capabilities of the models related to WAV files in regards to producing music-resembling audio. Since music taste is rather subjective, it will be a challenge to determine what type of samples are generally preferred. In this section, we will highlight the different tasks we will perform in order to provide an objective judgment.

9.1 Sampling music

The first step is the production of music samples. For each model, we will generate a total of 10 music files of 10 seconds. 5 files are generated using the techno test set, while the other 5 are generated using the classical test set. Similar to the data preparation procedures (see Section 7), most of the code related to sampling music was pre-written. The procedures, therefore, partly depend on the programmers their preferences. It is for this reason that we dedicate two different sections to the sampling of music: one for Wavenet and one for SampleRNN.

9.1.1 Sampling using Wavenet

Comparable to our procedure in Section 5.1, we sample a vector v_0 from a test set as starting point. The constructed Wavenet model will, at this moment, estimate probabilities using a softmax layer with $T = 1.0$. This results in S vectors of length 256. However, we will only consider the last vector of probabilities, because this is the only vector that gives information about a new time point. After the vector $\{\hat{y}^c_{t+1} : 1 \leq c \leq \mu + 1\}$ is estimated, $\hat{w}_0$ is sampled accordingly. Here, $\hat{w}_0$ stands for the integer representations of the quantized values at time point $t = 0$. Hence, we need to transform this value to 16-bits. μ-law quantization is reversed by using Equations 31 and 32. After which $\hat{w}^{''}_0$ is saved in an output list.

$$\hat{w}^{'}_t = 2 \cdot \frac{\hat{w}_t}{\mu} - 1 \tag{31}$$

$$\hat{w}^{''}_t = sign(\hat{w}^{'}_t) \cdot \frac{1}{\mu} \cdot ((1 + \mu)^{|\hat{w}^{'}_t|} - 1) \tag{32}$$

Finally, we determine the values for v_1. Every time we create a new input vector, $v^{l-1}_{t+1} = v^l_t$ (for $l = 2, ..., S$) and v^S_{t+1} equals the probabilities $\{\hat{y}^c_{t+1} : 1 \leq c \leq \mu + 1\}$. This process is repeated until we have 10 seconds of sounds, equivalent to 160,000 iterations.

9.1.2 Sampling using SampleRNN

Unlike the previous sample procedures we discussed, SampleRNN uses 3 vectors as input, each related to their own time clock. Which we refer to as v_{t_1}, z_{t_2} and

g_{t_3} for tier 1, 2 and 3, respectively. t_1, t_2 and t_3 represent the tier-dependent time points (see Section 8.2). g_{t_3} passes through the RNNs on the top module to produce the output information $c^3_{(t_3-1)*r^3+0}, ..., c^3_{(t_3-1)*r^3+r^3-1}$ (See Equation 26). Tier 2 uses the resulting information and z_{t_2} to produce $c^2_{(t_2-1)*r^2+0}$, $..., c^2_{(t_2-1)*r^2+r^2-1}$ on its turn. A softmax layer ($T = 1$) considers v_{t_1} and the information of module 2 to produce $\{\hat{y}^c_{t_1+1} : 1 \leq c \leq \triangle\}$ (see Section 8.2.1). $\hat{w}_{t1}$ is sampled accordingly, but is still in its quantized form and needs to be converted to 16-bits. The process of reversing linear quantization is given in Equation 33. Finally, the resulting value $\hat{w}^{'}_{t1}$ is added to the current output list, where $-1 \leq \hat{w}^{'}_{t1} \leq 1$.

$$\hat{w}^{'}_{t1} = \frac{\hat{w}_{t1}}{\frac{1}{2} \cdot \triangle} - 1 \tag{33}$$

v_{t_1} equals the set of previous values $\{x_{t_1-(FS^1-1)}, ..., x_{t_1}\}$. While z_{t_2} equals $\{x_a, x_a, ..., x_{a+(FS^2-1)}\}$ and g_{t_3} exists of $\{x_b, x_{b+1}, ..., x_{b+(\prod^3_{i=2} FS^i-1)}\}$. At the end of every iteration, we set $v^{l-1}_{t_1+1} = v^l_{t_1}$ (for $l = 2, ..., FS^1$) and $v^{FS^1}_{t_1+1} = \hat{w}^{'}_{t_1}$. z_{t_2} is only updated after FS^2 iterations of tier 1. The new frame z_{t_2+1} considers the values $\{x_{a+FS^2}, x_{a+FS^2+1}, ..., x_{a+2\cdot FS^2-1}\}$. Similarly, g_{t_3} is updated after $\prod^3_{i=2} FS^i$ iterations of tier 1, where g_{t_3+1} equals $\{x_{b+\prod^3_{i=2} FS^i}$, $x_{b+\prod^3_{i=2} FS^i+1}, ..., x_{b+2\cdot\prod^3_{i=2} FS^1-1}\}$

9.2 Listening session II

The generated samples from Section 9.1 will be used to conduct two experiments. Similar to the set-up described in Section 5.2, we will first see if we can draw obvious conclusions ourselves. If this is not the case, we will gather a group of volunteers to give their judgment about the audio files. In addition, we will make pairs between two sets, where each set provides a counterpart. The listeners will be asked which of the two files in the pair sounds more like music or whether they have no preference. Ultimately, we will determine the overall preferences towards one of the two sets. Comparisons between counterparts of pairs are given by Equations 16 and 17. The preferences towards music sets are determined by Equation 18. Finally, the volunteers will provide an overall ACR score (see Section 5.2) for the two sets and a short explanation for this score.

Notice that we have 4 types of WAV files to our disposal and there are 5 files of each type. Wavenet and SampleRNN were both used to generate 10 sample. Both models used the techno test set to produce 5 samples and the other 5 samples were produced using the techno test set. The first experiment we will conduct relates to the sub-question: *"Which state-of-the-art model is more capable of learning temporal relationships based on waveforms and generate music-resembling audio based on these relationships?"*. The two sets consists of the 10 samples produced by Wavenet and SampleRNN, respectively.

Furthermore, the second experiment we will conduct relates to the sub-question: *"Does the type of dataset influence the learning capabilities of models*

in terms of capturing patterns in music and producing music-resembling audio accordingly?". The sets of music samples that are used consist of all 10 samples that were generated using the techno and classical test set, respectively.

Finally, the results of both sets help us to answer the following sub-question as well: *"Do neural networks with lower loss values also produce more music-resembling audio?"*.

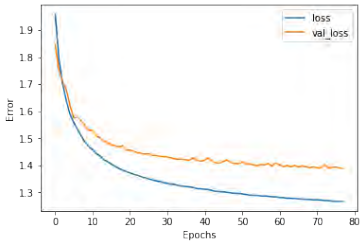

Figure 40: Wavenet's training results on classical music

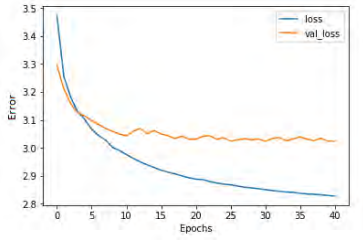

Figure 41: Wavenet's training results on techno music

10 Results & Evaluation

This section highlights the results of the models related to the WAV files. First, we will look at Wavenet and SampleRNN their ability to learn patterns in techno and classical music. After judging their learning capabilities, we will analyze the quality of their resulting music samples.

10.1 Results Wavenet

Figures 40 and 41 show the training developments of Wavenet for the classical and techno dataset, respectively. Moreover, Table 2 summarizes the final losses for the relevant training, validation and test sets. We can observe that Wavenet is better capable of analyzing the classical music files. The training and validation losses of the classical dataset are already lower than the final losses of the techno datasets after just one epoch. This means the classical music files are better suitable for learning patterns that are more generalizable. Furthermore, we can observe that early stopping terminates the training for both genres. The model instance trained on techno music already stops after 41 epochs, while the classical instance trains for 77 epochs. This further strengthens our notion of Wavenet having more trouble learning techno music.

Table 2: Results Wavenet

	training	validation	test
Classic	1.27	1.39	1.16
Techno	2.83	3.02	2.93

10.2 Results SampleRNN

In Figures 42 and 43, we can see the developments of the two SampleRNN implementations. Figure 42 summarizes the losses of the instance trained on classical music, while 43 summarizes the losses of techno instance. Furthermore, the final losses of the related sets of audio files are shown in Table 3. Similarly

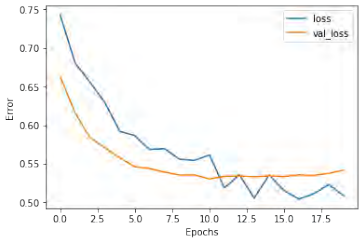

Figure 42: SampleRNN's training results on classical music

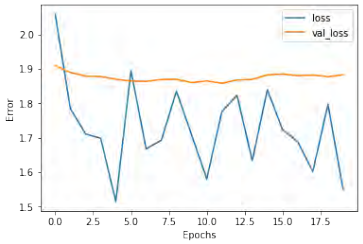

Figure 43: SampleRNN's training results on techno music

to Wavenet (see Table 2), we can observe that SampleRNN performs better when analyzing classical music samples. This is because the final losses of the training, validation and most importantly the test set are significantly lower. Furthermore, we can see that SampleRNN can find generalizable patterns in classical music for at least the first ten epochs. As its validation loss is decreasing during this time. After 10 epochs, however, the classical music instance seems to slightly overfit, as the validation loss is slowly increasing. The techno instance, on the other hand, does not find any relevant pattern after the first epoch. We can observe this by the fact the validation loss stays stable during training. In addition, there seems to be some periodicity in the training loss, although the related trend does not seem to increase nor decrease.

Table 3: Results SampleRNN

	training	validation	test
Classic	0.45	0.54	0.60
Techno	1.54	1.88	2.32

10.3 Wavenet versus SampleRNN

By looking at Tables 2 and 3, we can compare the performances of Wavenet and SampleRNN. For every subset (e.g., classical training set, classical validation set, techno test set, etc.), SampleRNN has a lower loss. That means our Samplernn instances are beter capable of finding patterns based on waveforms.

10.4 Listening session II

In this section, we will analyze the quality of the music samples produced by the waveform-based models. As stated in Section 9.2, we have 4 different types of WAV files to our disposal. We implemented two models, and two instances of each model. The first instance was trained on classical music and the other one techno music. Moreover, we have 5 files of 10 second of each kind. In the first experiment, the 10 samples produced by Wavenet will be compared with the 10

samples produced by SampleRNN. Additionally, in the second experiment, we compare samples based on the classical music set with the ones based on the techno music set.

Before we gather volunteers, we first compare the related audio file sets of both experiments. After listening to the music samples, we conclude that none of the four sets contain audio that even slightly resembles music. The files generally sound noisy and random. We, therefore, decide not to proceed with the two listening experiments, as we do not expect them to provide us valuable insights.

11 Conclusion & Discussion

In order to answer our main research question *"What is the current state of deep neural networks in modelling and generating music-resembling audio?"*, we have stated four sub-questions (see Section 1). In this section, we will answer the sub-questions one by one and highlight the limitations and shortcomings of our research.

The first sub-question relates to the analyses of notes and tempo and is stated as: *"Are deep neural networks capable of learning temporal relationships based on notes and tempo and generate music-resembling audio based on these relationships?"*. If we look at the Figures 9, 10 and 11, we can see that all MIDI NNs have decreasing validation losses for dozens of epochs. Hence, they are able to detect generalizable temporal relationships based on notes and tempo. The strength of the related NNs, however, can vary per model. LSTM-I, for example, performs worst in predicting offsetdiffs (see Figure 12), but best in categorizing notes and chords (see Figure 13). Furthermore, by looking at Table 1, we can see that all MIDI NNs have lower test losses than the Markov model. Thus, MIDI NNs are able to detect more intricate patterns next to the most simple ones. This can also be heard in the produced music. Music samples from MIDI NNs tend to be preferred over samples of the Markov chain. Furthermore, MIDI NNs have a ACR score of approximately 3 and this is higher than the ACR score of the Markov chain (see Figure 20). Lastly, after translating the ACR score, we conclude that the samples of the MIDI NNs are of resonable quality.

In addition, we will discuss SampleRNN and Wavenet their ability to analyze raw audio waveforms. This will be achieved by answering the following sub-question: *"Which state-of-the-art model is more capable of learning temporal relationships based on waveforms and generate music-resembling audio based on these relationships?"*. When we compare Tables 2 and 3, we can see that SampleRNN has a lower loss for every sub-dataset. Thus, SampleRNN is better capable of learning generalizable temporal relationships based on waveforms. However, there is no significant difference in the perceived music quality. A possible cause is SampleRNN's losses not being low enough.

Both state-of-the-art models were trained on two datasets, namely: a classical one and a techno one. The third sub-question relates to the comparison of the use of the two datasets and is, therefore, stated as: *"Does the type of dataset influence the learning capabilities of models in terms of capturing patterns in music and producing music-resembling audio accordingly?"*. If we compare the losses of the classical and techno datasets per model individually (Tables 2 and 3), we can see that all techno losses are significantly higher. This means that the models are less capable of finding generalizable patterns in the techno dataset. Thus, the type of dataset does influence a model's learning capabilities significantly. However, there does not seem to be a significant difference in the perceived quality of the music samples. Again, this is possibly caused by the classical losses not being low enough.

The last sub-question of our research is *"Do neural networks with lower*

loss values also produce more music-resembling audio?". This seems the case when comparing the MIDI NNs with the Markov chain. However, there are a number of findings that contradict this notion. Firstly, we proved that LSTM-II produces music samples of less quality compared to LSTM-I and CNN, although its losses were lower. Secondly, we saw that there is no significant difference between the quality of the audio files of SampleRNN and Wavenet. Finally, although the techno dataset is harder to analyze, there is no noticable difference in the produced music samples compared to produced classical samples.

However, a possible reason of us not finding a relationship between losses and music quality could lie in the choices we made. During the comparison of the MIDI NNs, we used a loss function that was the average of the losses of two features: notes and chords and offsetdiffs. We assumed that minimizing this function would lead to better music. However, Figures 26 and 27 indicate that predicting notes and chords is more important than predicting offsetdiffs. To examine their individual importance more extensively, one can perform experiments where music is produced using different weights for the two losses of the features. This way, one can be more certain of a relationship between lower losses and higher musical quality.

Moreover, although we applied deep learning during our research and were able to successfully constructs models with millions of parameters, we were limited by our computer power. More efficient models or more computer power could give us the possibility to examine NNs more extensively. We could, for example, perform more extensive parameter optimization, implement deeper models, run models for more epochs, among other things. Hence, this could have possibly led to more insights or to the production of more qualitative music. Van den Oord et al. (2016c), for example, concluded that using Wavenet with a receptive field of several seconds is crucial for producing qualitative music, which was not possible within our capacities.

In addition, during the listening session, the volunteers knew which samples belonged to a certain set. Although they did not know what the exact differences were between samples from different sets, they could still have had an unconscious biased towards a set. The biases of the volunteers could have influenced the results of the listening session, although we did not examine this.

Lastly, we did not work with an independent expert on music theory. Someone with sufficient knowledge on the matter could have influenced our research positively. During the listening sessions experiments, for example, we asked the volunteers if the music sounded well according to them. However, the quality of music is rather subjective and can, hence, differ per person. A domain expert could judge to what extent a NN is capable of recognizing patterns that are in line with music theory. Thus, he or she would be able to give more objective judgments.

12 Future work

In this section, we will discuss possible extensions to our work. In particular, we will highlight subjects we would have liked to investigate, but frankly did not have the time or resources for.

Firstly, during the data preparation procedure for the MIDI NNs (see Section 3), we only considered music that was played on a piano. An extension to our research, could be the analysis and production of MIDI files consisting of multiple instruments. This could be established by extending the number of features per time step from two to three by adding information about the type of instrument.

Secondly, we saw that the type of dataset has influence on models learning patterns in waveforms (see Tables 2 and 3). However, we have not investigated what makes datasets more suitable for training. We, for example, do not know if the classical dataset was easier to analyze because it only concerned one instrument or because of the characteristics of classical music. An extension to our research would, therefore, be the investigation of the suitability of instruments and music genres for training. The resulting insights could lead to samples of reasonable quality that are produced by waveform-based models.

Furthermore, we only used three models to analyze patterns based on notes and tempo and two models to analyze waveforms. To gain more insight about the current learning capabilities of NNs, one could implement more models. This way, more conclusions can be made about what type of models performs better in capturing relationship in music (e.g., RNNs versus CNNs).

As explained in Section 11, our lack of computational power prevented us from increasing the complexity of our models. A possible extension to our research is to use models similar to the ones we implemented, but with more complexity. For example, by extending the number of convulational layers in our CNN model (Section 4.2.2), or the depth of our Wavenet implementation (Section 8.1). More complex or deeper model increase the learning capabilities of models and could, therefore, lead to music of better quality.

References

[1] Boulanger. *The Csound book: perspectives in software synthesis, sound design, signal processing, and programming*. 2000.

[2] Briot et al. *Deep learning techniques for music generation-a survey*. 2017.

[3] Campbell & Swinscow. *Statistics at Square One*. 2016.

[4] Cho et al. *Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation*. 2014.

[5] Chung et al. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". In: (2014).

[6] Dannenberg. *Nyquist, a language for composition ands ound synthesis*. 1997.

[7] Engel et al. "Nsyth". In: (2017).

[8] Hadjeres & Pachet. *Deepbach: a steerable model for bach chorales generation*. 2016.

[9] He et al. *Deep Residual Learning for Image Recognition*. 2018.

[10] Hiller & Isaacson. *Experimental music; composition with an electronic computer*. 1959.

[11] Hochreiter & Schmidhuber. "Long short-term memory". In: (1997).

[12] Kaiser & Sutskever. *Neural gpus learn algorithm*. 2015.

[13] Kalchbrenner et al. *Grid long short-term memory*. 2015.

[14] Kingma & Ba. *Adam: A method for stochastic optimization*. 2014.

[15] LeCunn & Yann. *LeNet-5 convolutional neural networks*. 2013.

[16] Markov. *Rasprostranenie zakona bol'shih chisel na velichiny, zavisyaschie drug ot druga*. 1906.

[17] McCartney. *Supercollider: anew real time synthesis language*. 1996.

[18] Mehri et al. "SampleRNN: An Unconditional End-to-End Neural Audio Generation Model". In: *arXiv preprint arXiv:1612.07837* (2016).

[19] Mor et al. "A Universal Music Translation Network". In: (2018).

[20] Ravanelli & Bengio. *SPEAKER RECOGNITION FROM RAW WAVEFORM WITH SINCNET*. 2018.

[21] Shapiro & Wilk. *An analysis of variance test for normality (complete samples)*. 1965.

[22] Shuqi et al. "Music Style Transfer: A Position Paper". In: (2018).

[23] Srivastava et al. *Training very deep networks*. 2015.

[24] Usman. *Power Efficiency of Sign Test and Wilcoxon Signed Rank Test Relative to T-Test*. 2015.

[25] van den Oord et al. “Conditional Image Generation with PixelCNN Decoders”. In: (2016b).

[26] van den Oord et al. “Pixel Recurrent Neural Networks”. In: (2016a).

[27] van den Oord et al. “WAVENET: A GENERATIVE MODEL FOR RAW AUDIO”. In: (2016c).

[28] Yanchenko. *Classical Music Composition Using Hidden Markov Models*. 2017.

[29] Yu & Koltun. “Multi-scale context aggregation by dilated convolutions”. In: (2016).