DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2019

MahlerNet

Unbounded Orchestral Music with Neural Networks

ELIAS LOUSSEIEF

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

MahlerNet

Unbounded Orchestral Music with Neural Networks

Swedish subtitle:
Orkestermusik utan begränsning med neurala nätverk

Master's Thesis in Computer Science
KTH Royal Institute of Technology, Sweden
School of Electrical Engineering and Computer Science

By: Elias Lousseief

Date: *June 28, 2019*
Course: *DA225X*
Supervisor: *Bob L. Sturm, bobs@kth.se*
Examiner: *Hedvig Kjellström, hedvig@kth.se*

Abstract

Modelling music with mathematical and statistical methods in general, and with neural networks in particular, has a long history and has been well explored in the last decades. Exactly when the first attempt at strictly systematic music took place is hard to say; some would say in the days of Mozart, others would say even earlier, but it is safe to say that the field of algorithmic composition has a long history. Even though composers have always had structure and rules as part of the writing process, implicitly or explicitly, following rules at a stricter level was well investigated in the middle of the 20th century at which point also the first music writing computer program based on mathematics was implemented.

This work in computer science focuses on the history of musical composition with computers, also known as algorithmic composition, using machine learning and neural networks and consists of two parts: a literature survey covering in-depth the last decades in the field from which is drawn inspiration and experience to construct MahlerNet, a neural network based on the previous architectures MusicVAE, BALSTM, PerformanceRNN and BachProp, capable of modelling polyphonic symbolic music with up to 23 instruments. MahlerNet is a new architecture that uses a custom preprocessor with musical heuristics to normalize and filter the input and output files in MIDI format into a data representation that it uses for processing. MahlerNet, and its preprocessor, was written altogether for this project and produces music that clearly shows musical characteristics reminiscent of the data it was trained on, with some long-term structure, albeit not in the form of motives and themes.

Keywords: music, composition, algorithmic composition, neural networks, recurrent neural networks, RNN, variational autoencoder, VAE, LSTM, BALSTM, MusicVAE, PerformanceRNN, BachProp

Sammanfattning

Matematik och statistik i allmänhet, och maskininlärning och neurala nätverk i synnerhet, har sedan långt tillbaka använts för att modellera musik med en utveckling som kulminerat under de senaste decennierna. Exakt vid vilken historisk tidpunkt som musikalisk komposition för första gången tillämpades med strikt systematiska regler är svårt att säga; vissa skulle hävda att det skedde under Mozarts dagar, andra att det skedde redan långt tidigare. Oavsett vilket, innebär det att systematisk komposition är en företeelse med lång historia. Även om kompositörer i alla tider följt strukturer och regler, medvetet eller ej, som en del av kompositionsprocessen började man under 1900-talets mitt att göra detta i högre utsträckning och det var också då som de första programmen för musikalisk komposition, baserade på matematik, kom till.

Den här uppsatsen i datateknik behandlar hur musik historiskt har komponerats med hjälp av datorer, ett område som också är känt som algoritmisk komposition. Uppsatsens fokus ligger på användning av maskininlärning och neurala nätverk och består av två delar: en litteraturstudie som i hög detalj behandlar utvecklingen under de senaste decennierna från vilken tas inspiration och erfarenheter för att konstruera MahlerNet, ett neuralt nätverk baserat på de tidigare modellerna MusicVAE, BALSTM, PerformanceRNN och BachProp. MahlerNet kan modellera polyfon musik med upp till 23 instrument och är en ny arkitektur som kommer tillsammans med en egen preprocessor som använder heuristiker från musikteori för att normalisera och filtrera data i MIDI-format till en intern representation. MahlerNet, och dess preprocessor, är helt och hållet implementerade för detta arbete och kan komponera musik som tydligt uppvisar egenskaper från den musik som nätverket tränats på. En viss kontinuitet finns i den skapade musiken även om det inte är i form av konkreta teman och motiv.

Nyckelord: musik, komposition, algoritmisk komposition, neurala nätverk, recurrent neural networks, RNN, variational autoencoder, VAE, LSTM, BALSTM, MusicVAE, PerformanceRNN, BachProp

Preface

The subject of computers and arts have a long imaginative history and has been brought up in a wide range of fields including philosophy, entertainment, arts and technology. As a music academic and a person who is generally interested both in discussions and arts, I have often been asked if a computer will ever be as good a composer as a human. The anticipated answer has not always been the same and while some feel that music is just a schematic system of musical events that about any decent artificial intelligence should be able to jot down, others appreciate that music has a vast depth and amounts to one of those few things that are utterly and genuinely human and farther away than everything from being synthesized by machines. My position has always been that I believe that computers won't be able to come up with anything that a human doesn't instruct it to come up with and then, indirectly, it is a human who is the creator and not the machine.

When I first saw the results from Google's Deep Dream Generator project, it was the first time ever that I felt that a machine had created something artistic in an original and autonomous way. Finally, a machine had done it! Given my past as a composition student, I was intrigued with what neural networks could do with music, and at that moment, the seed to an idea of a master's thesis topic was planted.

This thesis is the realization of a neural network capable of modelling music. Besides the inspiration from Google's Deep Dream Generator project, an important goal with this master's thesis has been to learn a low-level machine learning framework in depth. A final inspiration, not realized entirely in this thesis, has been the way neural networks lend themselves to conditioning; a dream that has kept me going has been the idea of a neural network's understanding of the fusion of my big hero Gustav Mahler and some famous pop artist, for example Madonna or Robbie Williams.

The reference system used in this thesis is APA-like, which in reality means APA with two changes. First, all references use a bold and italicized highlighting to make it easy for the reader to skip large chunks of references and continue reading the actual text. Second, a reference can have a parenthesis with a model name. This has been added to make it easy for readers to follow specific models and put facts given about a certain reference into a context. Thus, the more condensed reference style, with "[1-32]" for example, was not an option since it would have made this sort of tracking of individual models impossible without looking at the references section, in the back, constantly.

Enjoy the reading!

Elias Lousseief, Stockholm, May 15 2019

“Somewhere underneath,

very deeply,

there’s a common place in our spirit where the beauty of mathematics and the beauty of music meet.

But they don’t meet on the level of algorithms or making music by calculation.

It’s much lower, much deeper,
or much higher you could say.”

Györgi Ligeti ***(Edwards, 2011)***

Contents

1 INTRODUCTION

1.1 Background

Ever since the dawn of computers, it has been a standing question whether these machines can produce arts or not. As humans, perhaps feeling threatened by the superiority of these machines on some tasks, we have often emphasized that arts is something that is reserved to the human mind only; a last privilege in a rapidly more and more automatized world. Arts performers, however, are often faced with the question of how to renew themselves and how to not get stuck in old and familiar patterns. Throughout the years, in all arts fields, diverse sets of systematized techniques have been used to come up with new themes and expressions and a well-known principle in aesthetics is that you need to learn the rules of the old masters in order to truly break them. With this in mind, wouldn't it be fantastic if we could use the unbiased "minds" of computers to head up in new directions without being affected by our own previous inspiration and habits?

In the last decade, a sub-field of machine learning focused on arts has emerged and there are now machine learning communities that are active in virtually all arts fields where the art can be quantified as data. Due to that image recognition and processing have always been popular tasks and data in these fields exist in vast quantities, it is not surprising that arts dealing with images have been explored more than others, with results reaching new heights with the arrival of the Deep Dream Generator ***(Mordvintsev et al., 2015)*** which can actually create astonishing image art that bears the impression of being from another world. Nevertheless, other arts fields have also been explored. Compared to classic machine learning problems, results in the arts field have, as expected, been much harder to interpret and evaluate in a unified way, as is the case with art in general.

The method of creating music with the help of programming and computers has been given the expression *algorithmic composition* and is by no means limited to music created with machine learning. Quite to the contrary, different categories of algorithms, including deterministic, genetic and random algorithms, have been tested for this purpose ever since the 1950's with varying purposes and results. During the recent machine learning revolution, much work has been invested in teaching machine learning algorithms to compose music resulting in a very diverse field with results pointing in many directions. As with contemporary machine learning and artificial intelligence in general, a large portion of these attempts have involved artificial neural networks.

There are many reasons to why investigating machine learning techniques for arts, and more specifically music, is interesting. These, in turn, depend on the intended use of these techniques, which is not something that everyone agrees on in the community today. Given the commercial point of view of a fully automated music composition system, there are numerous places and occasions where original music is needed for some purpose. For example in video games, in movies and in clubs where music is a large part of the focus, automated music, perhaps conditioned on a particular style, would indeed fill a purpose, perhaps intimately associated with demise for today's composers. Also in places where music is less in focus, for example in elevators and in lounges, automated music production would fill a role and most certainly be cheaper to play than music composed by humans with the added benefit of being able to generate new and unique music at all times. With less commercial purposes in mind, a music composition tool, fully automated or requiring human interaction, could act as an aid to composers helping them to develop and get new ideas. Finally, there lies an aesthetic-scientific interest in finding out what latent factors a program can track down and steer, given a set of historical music pieces and styles.

1.2 Problem Statement and Limitations

Given both the current main focus of machine learning in general, but also recent streams of algorithmic composition in particular, the overall purpose of this work is to construct, train and evaluate a novel neural network architecture capable of composing music in the same style, or a fusion thereof, as the music it was trained on.

At a higher level of detail, this implies an objective that is two-fold:

- *What does the history of algorithmic composition in general, and with neural networks in particular, look like?* This involves researching the field of algorithmic composition, focusing on recent algorithmic composition in machine learning in general, and with neural networks in particular, and collect the research in a literature survey.
- *Considering recent advancements, what would a new neural network architecture for algorithmic composition look like and how can it be implemented?* This objective implies the design, implementation, training and evaluation of a new neural network architecture based on the literature survey from the first objective.

In particular, the designed neural network should fulfill the following conditions (C):

C.1 Model polyphonic music.

C.2 Model a variable number of different instruments.

C.3 Be steerable by means of seeding or conditioning to some degree, in the way that it outputs different compositions as a result of different seeds and / or conditionings.

C.4 Have no restrictions on the length of the output music.

C.5 Be end-to-end and not require any extra manual annotations.

C.6 Make minimal assumptions on the input.

C.7 Model symbolic (notated, as opposed to sounding) music in its basic form only, that is, a stream of notes against an underlying meter possible to visualize in a rudimentary score. Tempo, dynamics, accents, character and other playing instructions are out of focus as well as any sounding musical properties such as micro-timing or ornaments.

C.8 Model pitch-based instruments only and not purely rhythmic instruments.

The rest of this thesis has the following structure: in section 2 some theoretical background on music as well as machine learning and neural networks is presented and then section 3 is the literature survey, also filling the role of a previous or related work section. Both of these sections can be omitted for readers who are familiar with the theory of music, machine learning and algorithmic composition in particular. At the end of the literature survey, there is a brief summary of the most common paths that have been taken with respect to different aspects historically. MahlerNet and justifications for design choices taken during its implementation are presented in section 4 after which comes section 5 where three different experiments and results are presented without further interpretation. In section 6, the results in section 5 are commented, elaborated on and analyzed and section 8 contains a short summary of what has been done and what conclusions can be drawn from it. Section 7 contains directions of interest and paths for future exploration, both with this work in mind and in general. After the bibliography follow appendices containing tables gathered during the literature survey as well as details on different aspects from section 5.

1.3 Ethical, Societal and Sustainability Aspects

As always with machine learning and artificial intelligence, it is hard to know what overall impact the contribution within a very confined field may have when ported to other fields. Numerous are the historical occasions when important, sometimes horrible, inventions arrive in the form of something else where the world, or the inventor herself, might have not anticipated the chain of events to follow. In the world of artificial intelligence (AI), there exists groups and organizations that are *for* an open source climate, but that still don't make available their source code because they think it is to early to say what it might be used for. Boards of scientists have formed, for example the Institute of Life, warning against taking the threat of artificial intelligence too lightly and advising that more money is spent on investigating the effects and dangers of AI which currently is a question with much less economic interest than has the development of AI itself.

To conclude, one might never know what algorithms and machine learning models may be used for, and given that this work is about music, it is evident that there is no current malevolent purpose and so any use of algorithms in such a way would be a large demise. However, by using contributions from previous works and designing an architecture with specifically music in mind, the work is as specialized as possible, and therefore less likely to be used in a negative way than a more general model of some sort. With this in mind, the source code will be made available to the public with the motivation that similar models are already available and the novel contributions of this work is mostly specific to music.

Automatically generated music is of interest in a lot of places, for example in movies, video games, elevators and lounges. Aesthetically, music modelling could also leads us in new directions and give inspiration to head in new ways that would otherwise be unexplored, as well as help us understand the old masters from a new perspective. A negative aspect then deals with how artificial intelligence potentially can make some professions obsolete, leading to unemployment, or harder working conditions, for large groups of people. Composers may not be a large profession group but most of them likely live under quite difficult circumstances where it is already today hard to get work and a steady income. However, on a theoretical level, it is not right to stop the development just to save some groups of professions; it has not worked historically and it will not work in the future. One might also hide behind a classic argument and state that no matter if this work makes a progress or not, someone else's will. Finally, one might also employ a more positive outlook on things and adopt the view that professions don't disappear, they just change form. Alas, as we shall see, the development of neural networks for musical composition gives rise to a new area of creative work, namely the design of neural networks for music composition.

2 FUNDAMENTALS

This section contains brief theory on music and machine learning with neural networks included for the purpose of making vocabulary and principles in the rest of this work, especially in the literature survey, more available to the reader in need of it. Readers who are familiar with both music theory and neural networks can skip it.

2.1 Music

Everyone is familiar with music but most people do not know music in a formal way. Music can be considered a vertical and horizontal stream of events where the vertical dimension corresponds to different voices, sometimes but not always equivalent to instruments, playing at the same time (one-dimensional in case of a melody only), and the horizontal dimension corresponding to how the music unfolds in time. Statistically, with respect to the horizontal dimension, later events are typically conditioned on earlier, that is, we hear repetitions, variations and can keep track of what the music sounds like based on what we have heard before; a sort of abstract story one might say. In the vertical dimension, conditioning occurs as well, this being a slightly more complicated subject which we will derive in this section.

The simplest musical events are in fact, when written down, called *notes*. A note has several properties: *pitch* (height), *duration* (for how long it sounds), *timbre* (instrument or characteristic of the sound), *dynamics* (strong or soft), *envelope* (how it unfolds during its duration), *timing* (property of the note when it is played live; does it start exactly on time or a little late or early?) and more. In symbolic music, most of this info is written in the music even though there is room for personal interpretation as well. The height of a note is called *pitch*, and it can be given in the unit Hz, even though we more often discuss pitch within the western tonal system that we have devised. Here, pitches are placed out in a logarithmic fashion (with respect to wavelength in Hz) at different intervals on a real axis measuring ascending Hz. The locations are called C, C# / Db, D, D# / Eb, E, F, F# / Gb, G, G# / Ab, A, A# / Bb, B (sometimes called H). These pitches could have been placed out at whichever intervals and it is by no means the case that the western tonal way of doing this is the only correct, even though there are reasons related to consonance and dissonance to why they are placed out this way. *Consonance* means that it is pleasant to hear one or a few notes, *dissonance* means that it is unpleasant. But what is meant by pleasant and unpleasant?

Physically, a sound wave is an oscillation with both amplitude and wavelength. The latter results in the height of the sound and the former in how loud the sound is. A note with a pitch with only one wavelength is called a sine wave. Those do not exist naturally but can be induced by synthesizers. In reality, sound waves interact and interfere so that a given natural sound really consists of many wavelengths together. When these wavelengths interfere in certain structured ways, we call it a pitch whereas when they interfere chaotically, we experience it as noise. Typically, we refer to the lack of interference between these waves as consonance and the presence of interference as dissonance (at an increasing scale of course; music contains a lot of dissonances but we still enjoy it). It turns out that a tone (the sounding result of a note), as we know it, is built up by a fundamental (from which the pitch of the note is derived) along with an infinite series of overtones, which are wavelengths that coexist with, and are divisible by (in terms of Hz) the fundamental creating the sound of a tone. Different instruments sound differently, have different timbres, because they emphasize different overtones. The overtones fulfill special mathematical properties relative to the fundamental (such as being doubles, triples etc...) and the higher up in the series you come, the tighter lie the overtones. This has an important implication for the anatomy of music.

To be correct, when we talked about the fixed locations on the real axis of Hz values, we really talked about *pitch classes*, not specific pitches, since there are several, for example, Cs to choose from. Not all pitches sounds similar, but some pitches sound very similar. These are the pitches where the other pitch has twice as many Hz as the first (for example, 2000 Hz, 4000 Hz and 8000 Hz) and they sound similar because the fundamental of the second is the first overtone in the overtone series of the first and the rest of their overtone series coincide to a high degree. These happen to have the same names in the western tonal system and we refer to all the pitches in such a series as a pitch class. In reality, the different pitches have specific names, such as C2, C1, C, c, c2, c3. There are different schools for how to name the pitches, some prefer to use C^{-2}, C^{-1}, C, C^1, C^2 and some people even call the pitch classes Do, Re, Mi, Fa, Sol, La, Si (Ti), Do according to a very old system called solmization. Due to a misinterpretation in the early Germany, B is sometimes referred to as H and Bb is referred to as B which can be very confusing. Other pairs of pitches sound slightly less similar because their overtone series do not coincide to the same degree but still to a quite high degree but maybe the fundamental itself is not an overtone of the other. The relationship between these pitches are the fundamental idea of the western tonal system which, in its infancy, favored pitches that were more or less similar in this manner, to achieve a high level of consonance. Over the years, the patience with the concept of consonance has been further and further expanded and today, we consider intervals that would have been considered utterly dissonant back in the Renaissance era, consonant. Actually, we even enjoy dissonance, but typically when it is in contrast to the very same notion of consonance that the western tonal system is built on. Thus, our western tonal system can be motivated by its bias toward physically motivated consonance.

Because of the overtone series, high pitches are conditioned on low pitches to a higher degree than low pitches are conditioned on high; almost all high overtones, if you go high enough, is included in any low sounding pitch whereas it sticks out as very upsetting to have a very low dissonance occurring as a result of two non-consonant base notes.

Each note also has a *duration*. In reality, in the interpretation of music, players often use so-called *micro-timing* and play according to a variable tempo, playing some notes longer than they should, some shorter, starting some earlier and others later. In the actual written music however, the duration of a note is very strict and even though, in theory, any length of a duration could be written down, there is a limit to how small note values will occur in a piece of music. Notes are furthermore arranged in bars and each bar has a number of underlying "silent" beats. We hear some actual beats as emphasized whereas others are not. All in all, this results in a so-called meter which is intimately associated with the number of beats and their emphasis in a bar. Another word for meter is time signature and the most common one is 4/4 whereas another one is 3/4, which is often used in waltzes. The second number is a duration whereas the first number indicates the number of such beats, or durations, in a bar (unit of music). When this number is 4, the first beat is the most emphasized followed by the third. The remaining beats are not emphasized at all. When that number is 3, the same applies even though it creates another feeling.

Music can be one stream of events where no two events happen at the same time, *monophony* (melody), or multiple independent streams of events with no restrictions on interactions, *polyphony*. Monophony with several accompanying streams which follow the same duration patterns and are subordinate to the melody is called *homophony*. Most western popular music is highly homophonic.

Given the pitch classes and pitches therein, music is further structured by *scales*, which indicate sets of pitches that fit well together, and that have been used throughout the history of western tonal music. The choice of pitches in a scale is ultimately related to the overtone series and also the way *chords*, multiple simultaneous pitches with equal durations, are formed. The fact is that the overtone series plays an important role on many hierarchical levels in music which is manifested in, for example, the *circle of fifths*, used to explain the harmonic, or chordal, progressions in western tonal music. A scale thus has associated with it a set of pitches as well as a set of chords that can be constructed from those pitches. During the course of a piece of music, one can smoothly change scale, or key as it is called, which is an action called *modulation*.

Two pitches have an interval in between them. The interval between any two consecutive Cs is called an *octave* and can also be determined by the number of half-steps (a half-step is the interval between two consecutive keys on a piano, both black and white keys included) that exist in between; 12. Not all scales need to have the same number of pitch classes and a scale with only half-steps contains 12 scale steps and is called a *chromatic scale*. Such a scale is not very common but the use of half-steps in general is common and is called *chromaticism*. A *whole tone* scale uses only whole steps and such a scale has 6 scale steps and is not very common either. The *pentatonic* scale has 5 scale steps and is often used in blues and jazz. The basic western tonal scales are called *diatonic* scales, because they have both half and whole steps in them, and have 7 steps. Two particular scales are called *major* and *minor* scales and differ in the distribution of half and whole steps; a major scale has a series of 1, 1, $\frac{1}{2}$, 1, 1, 1, $\frac{1}{2}$ steps from the start pitch whereas the minor scale has a series of 1, $\frac{1}{2}$, 1, 1, $\frac{1}{2}$, 1, 1 steps. If the scale wrap around, we can see the the major scale has the same distribution as the minor scale starting from the third step. The minor scale thus is related to the major scale on the third step in that they share pitch content. This is a common relationship in western tonal music and we say that A minor is the *relative minor* to C major and vice versa. A scale defines the base content of the music but it does not mean that we can not use any other pitch material at all; to the contrary, other non-scale pitches can be used to create contrast but the overall material and structure is traditionally from the scale.

All scales of a certain type have the same ordering of half and whole steps but begin on different pitches. Because not all instruments have the same range as a piano (especially not the voice) there are more than one major and minor scale; perhaps we need to shift the entire range of a melody a few steps up or down. Such an action is called *transposition* and it is often done for singers for example if the highest note in a given piece is too high; the piece is then transposed down until the highest note is within the range of the singer's voice, perhaps going from being in G major to E major. Most people can easily transpose on the fly when singing but the same most often requires new notated music when it comes to other instruments. The property of music being the same when transposed is called *transposition invariance* and the same does not go for languages. Consider the word "WORD". Transposing it one "step" up yields "XPSE" which does not mean the same thing to us as does "WORD".

Ultimately, the spacing between pitches in the western tonal music is a compromise between the overtone series (consonant intervals are the basis), physical limitations (one can't play a piano with 400 keys) and pitch density that allows for a slight room between consonant intervals. The tuning used in modern pianos is a compromise between all different keys (as in scales) that must be playable on the piano and earlier in history, there were tunings that resulted in a C major scale that sounded more consonant than it does today but with, in effect, an unplayable F$\#$ major.

Music is further put together according to schemes which are typical for different traditions. In western tonal music, which is the tradition that most of our western music today stems from, or at least is highly influenced of, tones form *melodies* and chords form progressions of *harmonies*. These are put together in *motives* which form *phrases* and *themes* which, in turn, form larger building blocks ***(Lousseief, 2015)***. Music thus has a highly hierarchical structure.

2.2 Neural Networks

Artificial Neural networks (ANNs or NNs), called neural networks informally, is a class of machine learning algorithms inspired from the human nervous system that were first thought of in the middle of the 20th century. A great deal of theory was laid forward early but throughout time, neural networks suffered from two big setbacks after which followed time when little or no interest was invested in them. The first problem arose early on when it was thought that a neural network could do no better than a linear discriminator. This was solved by adding more layers. The second setback became apparent in the 90's when it was believed that training neural networks was almost impossible. The answer to this problem was the backpropagation algorithm and the use of differentiable activation functions instead of the step function. Alas, both of the setbacks were remedied in some time and today, this large class of algorithms represent the state of the art in the machine learning field.

Artificial Neural Networks have many different forms but its base component is an abstraction of a neuron that outputs a value after having received weighed inputs from input data or other neurons. Imitated from the human nervous system, the neuron can also have a so-called *activation function*, a non-linear function, that introduces a non-linearity in the otherwise linear combination of inputs that would otherwise restrict expressiveness. Common activation functions are *tanh*, *sigmoid* and *ReLU* (Rectified Linear Unit). In the basic neural network, sets of neurons are placed in layers where all the neurons in one layer only receive weighed input from all the neurons from the previous layers. Such a network is called a *feedforward* network. In some specific networks (typically older ones such as *Hopfield* networks and *RBF* networks), there are connections between neurons in a layer but that is typically not the case in the vanilla network. A notable exception to this rule are *recurrent neural networks* (RNNs), used to model sequences whereby a recurrent state layer with self-connections is used to account for temporal context. The layer where the data is fed to a network is called the *input layer* and the layer where the output of the network is read is called the *output layer.* All layers in between are called *hidden layers* and may have more or less specialized functions depending on the type of network. As a general rule, the expressiveness of the network increases as more layers and neurons are added. A collective name for ANNs with more than one hidden layer is *deep neural networks* (DNNs).

The main components used to train a neural network are input data, *target data* (the desired output of the network that we train with), the network itself and a *loss function.* The loss function gives some measure of how well the network predicts the target data given the input data, with the goal that it should predict it well. Using the derivative of the loss function, the backpropagation algorithm iteratively updates the weights of the network which successively makes better and better predictions. In some scenarios, the network may *overfit*, which means that the network adapts too well to the actual training data and loses the ability to generalize what it has learned on other data. For this purpose, a *validation set*, not part of the training data, is used to test the model every now and then to see whether it still has its generalization capabilities or not. ANNs are usually trained with a number of training patterns in parallel, a *batch.* Batch sizes are typically powers of two and when all the training data has been processed, the training has gone through an *epoch.* For every batch there is a weight update which is why counting batches can also be referred to as counting *steps.*

The field of artificial neural networks is old but it has flourished in the last decades thanks to a series of progressions, for example in the form of more efficient algorithms, more available training data and better computing resources.

Now follows a list of more particular types of networks, or training setups, along with brief explanations.

- **Feedforward Neural Network (FFNN) or Multilayer Perceptron (MLP):**

  Feedforward networks might refer to both the category of neural networks that are not recurrent, and as such, may include a large variety of network architectures. However, feedforward networks may also refer to the simple, previously described, vanilla neural network, with feedforward layers with input, output and zero or more hidden layers in between, also known as multilayer perceptrons.

- **Simple Recurrent Network (SRN):**

  Simple recurrent networks are early RNNs with a simple formulation and were used before the equations of the RNN, as we know them, got entirely standardized. Typically, these networks are used liberally with custom configurations of fixed weights and modelling aspects built into the architecture. Two types of SRNs, Jordan and Elman networks, are used which differ only subtly. Both of them usually have three layers where the middle layer consists of hidden units. The output layer is connected to the input layer to provide for sequential data. In a Jordan network, the input layer has recurrent self-connections, providing context relative to the sequential nature of the data, whereas in an Elman network, the recurrent connections occur in the hidden layer. As with regular RNNs, *teacher forcing*, the principle where a recurrent network is fed target output values from the last time step instead of the actual output, is most often applied even though not always mentioned. SRNs were first trained by weight updates at every time step by comparing the target and output. Later, adaptations from the regular backpropagation algorithm brought about *backpropagation through time* (BPTT), the most common algorithm for recurrent networks in general, as well as alternative algorithms such as *real time recurrent learning* (RTRL) which is less common. Informally, there is no strict line between regular RNNs and Elman networks with recurrent connections in the hidden layer and with no time-typical customizations or inputs, there is no difference. What is implicated here is that an Elman network that only accepts as input the actual data sequences, where both the input layer and the output layer are fully connected with learnable weights to the hidden layer, is actually the same as what is later referred to as a vanilla RNN.

- **Recurrent Neural Network (RNN):**

  As with feedforward networks, recurrent neural networks (RNNs) is an expression that can be used to refer to several things. Its broadest use defines any neural network that models sequences and holds some sort of state as a result of this, in which case, an RNN can refer to an SRN, an LSTM (introduced below) or an actual architecture, being the most standard and regular among these recurrent networks. The RNN as a model is in essence a standardized SRN (more specifically an Elman network) which can be distinguished from an SRN in several aspects. First of all, it is said that SRNs have context units whereas RNNs have a hidden layer with a hidden state. Furthermore SRNs are often customized in the network itself, for example by partial connectivity between layers or fixed weights, in an attempt to guide the model whereas the plain RNN is assumed to have full connectivity between layers as all weights are learnable. Finally, SRNs are often conditioned with additional, non-learnable, inputs (plans) whereas by default, RNNs are typically trained with the input sequences only. Thus, the differences lies in idiomatic use and terminology rather than in the theory and, as is said in the section on SRNs, an Elman network without these additions is effectively the same as a vanilla RNN. A historical difference is also that SRNs are often illustrated with all neurons in each layer, clearly showing how recurrent connections work whereas RNNs (all sorts) tend to use the abstraction of one node with a recurrent arrow for such a layer which can sometimes cause confusion among beginners who mistake the node for one neuron only.

An RNN, as a model, thus has an input layer and a hidden layer. The RNN input layer is connected to the hidden layer which has recurrent connections and holds a hidden state (or context in SRN terms). At each time step, depending on what is modelled, the RNN may output data as well which is typically done by connecting the hidden layer to an output layer. In this type of modelling, the output from one time step is often the input to the next, but during training, teacher forcing is usually employed whereby the correct target is fed as input to the next time step instead of the one that was actually predicted during training. Teacher forcing is said to speed up learning, and works well given that the network learns to predict the right target eventually. However, if this doesn't happen, the performance of the network during training (with teacher forcing) and during inference (full reconstruction) may differ largely. This can be avoided in part by using curriculum learning strategies such as scheduled sampling where the teacher forcing is gradually replaced with next step inputs that are sampled from the network's own predictions ***(Bengio et al., 2015)***.

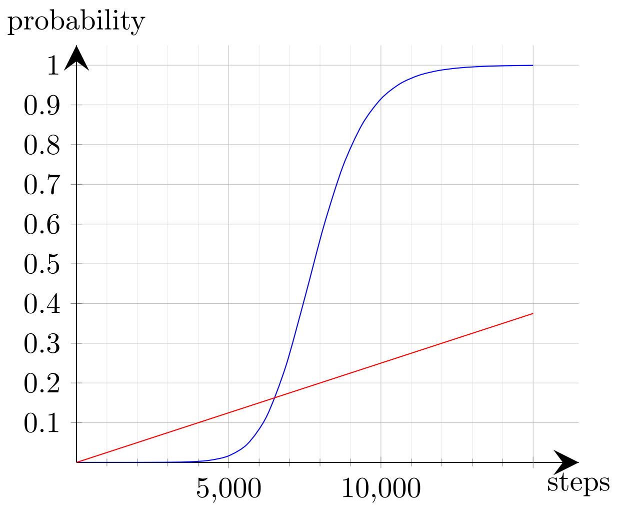


Figure 2.1: Probability of sampling from output predictions instead of taking the next input from the ground truth (teacher forcing) as a function of global training steps when using scheduled sampling. The blue graph shows an inverse sigmoid schedule with rate 1000 and the red graph shows a linear schedule with rate (linear coefficient) 0.000025

RNNs are typically used for continuous classification or regression, in which case the output corresponds to a class or a real value, or in a *sequence to sequence* (seq2seq) context, where the hidden state after the last input of a sequence is used as a summary of the entire sequence, and used as input for some subsequent process. RNNs are typically trained, as SRNs, with either backpropagation through time (BPTT), which can be thought of as unfolding the recurrent network over all its time steps into a multilayer feedforward network where the recurrent weights for the hidden layer are used in multiple layers, or real time recurrent learning (RTRL), which is less common.

- **Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU):**

  Both *long short-term memory* (LSTM) and *gated recurrent unit* (GRU) networks are like complex RNNs with gates that specifically handles the amount of context information to forget and the amount of input information to add along with a few other things. The LSTM maintains two contextual states, a cell state and a hidden state, as opposed to the GRU (and the RNN) that only maintains a hidden state. The LSTM was developed to remedy the problem of vanishing and exploding gradients, prevalent in RNNs keeping them from modelling long-term structure, and does so by adding portions of states to each other. The addition operator, being less vulnerable to the gradient problems than multiplication, is one ingredient to this. The LSTM is by no means the only way to extend RNNs with gates to remedy the gradient problem, even though the first, and much later, the simpler GRU was invented for the same reason. Also other solutions exist as well as variations on both the LSTM and the GRU. When comparing the LSTM to the GRU, the former is considered slightly more complicated which makes LSTMs slower to train. However, since LSTMs are more complicated, they should be able to express more complicated patterns. Even though this has not been completely shown, it has been shown that the LSTM, occasionally, manages to model slightly more complicated patterns than the GRU. This uncertain advantage along with the certain advantage of shorter learning phases has led to the GRU, not being very old as of this writing, growing in popularity. Learning in LSTMs, GRUs and friends is done in the same way as in RNNs and SRNs, and typically includes both teacher forcing and BPTT (or RTRL).

- **Restricted Boltzmann Machine (RBM):**

  A *restricted Boltzmann machine* (RBM) is an undirected network with two layers: one hidden and one visible. The visible layer holds a pattern and the hidden layer is used to discover latent features. The weights between the hidden layer and the visible layers are the same when sampling the hidden layer from the visible and vice versa. However, the two layers have different biases. Working with a trained RBM amounts to sampling the visible layer from the hidden layer according to some procedure. The model can give samples from a learned distribution when all the visible nodes are sampled from the randomly initialized hidden layer. Some visible nodes, however, may be clamped, so that their values are not changed during sampling, in which case the RBM does pattern completion or prediction according to the same distribution, given that the clamped units are the context history and the unclamped nodes are the next values to predict.

  RBMs stem from *Boltzmann machines* (BMs), which are very hard to train, and are restricted in the way that there is no interconnectivity between nodes in a layer, only between nodes in different layers. This restriction makes them easier to train since *Gibbs block sampling* can be used during training with *contrastive divergence* (CD), as opposed to regular BMs that have to be trained with single Gibbs sampling in a process based on *simulated annealing.* Extensions to contrastive divergence exists, such as *persistent contrastive divergence* (P-CD) which does not reset the state between learning iterations when Gibbs block sampling is performed on the hidden layer. Contrastive divergence usually comes with a parameter k (resulting in CD-k) indicating how many rounds of sampling are done before we consider ourselves satisfied with the sample. Most often this parameter is set to 1 and a single sampling iteration is performed, however, one should know that the sampling becomes more representative of the model's learnt distribution the higher values of the parameter k that is used, even though it also takes longer time. One way to solve this without any sacrifice is to use P-CD.

- **Deep Belief Network (DBN):**

  A *deep belief network* (DBN) is a hierarchy of RBMs where the hidden layer of a lower hierarchy RBM is the visible input of the higher hierarchy RBM. A DBN is trained greedily, that is, one layer at a time, starting with a single RBM. After this RBM is trained, a new RBM is added on top of it, taking the former's hidden layer as input and only the new RBM is trained. When the depth of the network is satisfactory and training is done, the entire net can be trained anew with small adjustments using gradient descent. DBNs often have *sigmoid belief networks* (SBNs) as higher hierarchy networks and an RBM only at the deepest layer. DBNs are used like trained RBMs, albeit with a higher representational power due to the number of layers.

- **Sigmoid Belief Net (SBN):**

  Sigmoid belief nets are similar to RBMs (and BMs) but different in that they are directed graphical models, as opposed to the latter. Originally, SBNs had hidden and visible nodes which were arranged in two parallel models: the recognition model consisting of the weights from visible nodes to hidden nodes, and the generating model, consisting of the weights from the hidden nodes to the visible nodes. The idea is that the network should be trained with visible patterns so that the generating model with high probability generates patterns similar to those that it has been trained on. This can be done in different ways but a common technique is the so-called *wake-sleep algorithm* which has two phases, much like the training algorithm for RBMs. In the wake phase, the recognition model is trained and in the sleep phase, the generating model is trained after which weight updates take place. SBNs are inspired from *Bayesian networks* with expert knowledge posing as hidden nodes and information (verdicts or conclusions) posing as visible nodes. Several versions of SBNs exist, such as the *fully visible sigmoid belief net* (FVBSN) which has no hidden nodes.

- **Neural Autoregressive Density Estimator (NADE):**

  A *neural autoregressive distribution estimator* (NADE) is a graphical directed model with a tractable density for which it is possible to calculate exact gradients. It is inspired from RBMs and FVSBNs and is an improvement over these architectures which have to be trained more or less heuristically. A NADE models a distribution according to some arbitrary ordering and models each variable conditioned on the previous ones ($P(x_t \mid x_1, x_2, x_3, ..., x_{t-1})$).

- **Autoencoder (AE):**

  Autoencoders have as training objective to reconstruct the input at the output and is, as opposed to most previous architectures, an example of *unsupervised* machine learning where no manually given targets are handed as part of the training procedure. Between the input and output layers sits one or several hidden layers where the innermost has fewer units than all other layers. This layer acts as a bottleneck and if the autoencoder manages to train properly, the contents of this layer, after input has reached it, acts as a compressed version of the input. Trained autoencoders can also denoise by supplying corrupted input or input with missing parts whereby the autoencoder can reconstruct the pattern closest to it. The dimension of the compressed code of the autoencoder is called its *latent space*, and it is highly unstructured, meaning that if we unhook the encoding part of the autoencoder and just sample from its latent space and reconstruct what we sample, latent vectors that are close to each other don't necessarily yield reconstructions that are similar.

- **Variational Autoencoder (VAE):**

  Being published independently by two different sets of authors, *variational autoencoders* (VAEs) were invented in 2014. A VAE is an autoencoder with its bottleneck replaced with a multivariate Gaussian distribution parameterized by the final outputs from the encoder layers. For reconstruction, a latent vector is sampled from this distribution and after training, the VAE has a latent space that is structured, unlike the autoencoder. A VAE is trained with both a reconstruction loss (the same as the standard autoencoder) and another loss that forces the latent space to be structured, acting as a regularizer on top of the regular training. This other loss is often called *KL loss* (from Kullback Leibler) since it is calculated from the KL divergence between the distribution of the sampled vector and a multivariate standard Gaussian. The two losses sometimes need to be weighed against each other depending on the problem context and the desired outcome, usually sacrificing the landscape of the latent space on behalf of the reconstruction quality. One way to weigh the KL loss is to use the *free bits* extension, whereby the KL loss is only taken into account whenever it is larger than a certain threshold ***(Kingma et al., 2016)***. β-VAE is another way, introducing the β weight that scales the KL loss, possibly according to some annealing scheme ***(Higgins et al., 2016)***. The idea with an annealing scheme is to let the VAE focus on reconstruction quality early in the training process, when training can go in many directions, and introduce the KL loss successively as training progresses.

  After training, sampling and reconstructing from the latent space yields samples from the distribution of the dataset it was trained on.

- **Convolutional Neural Network (CNN):**

In a *convolutional neural network* (CNN), there are convolutional layers in which a convolution, or a patch, with the number of weights as specified by the size of the convolution, is moved around the input yielding an output value for every position. The output is calculated from the dot product with the input at the position of the convolution, followed by a non-linearity. Each convolutional layer may have several convolutions, or *filters*, that processes the input and each outputs an output map, results in outputs with several *channels* if more than one filter is used. Each filter has its own weights and the reason why the depths is referred to as channels is because convolutional layers was initially used on 2D images where each data point was a pixel with depth 3, since there are 3 channels (R, G, B) per pixel.

- **Generative Adversarial Network (GAN):**

  A *generative adversarial network* (GAN) network is often illustrated by the notion of a game with two players where the first player, called the Generator, attempts to generate data that the other player, the Discriminator, cannot distinguish from real data. This game is maintained until the so-called *Nash equilibrium* is found and during training, the Discriminator is trained with both fake and real data and both models are updated successively until the Generator has reached a desired level. GAN networks are hard and unpredictable to train and much effort has been put into improving the training procedure. Some GAN networks have showed very impressive results and it is said that even though the VAE offers possibilities that GANs doesn't, the latter often produce better generative results that looks slightly more like the original data than the VAE does.

3 LITERATURE SURVEY

In this section, the history of algorithmic composition will be treated. A brief overview is first given from a historical perspective after which the focus will be shifted towards algorithmic composition on computers and more specifically with machine learning and neural networks whose development caught on in the 80's. At the end of this chapter, there is a brief summary that captures the most important aspects and streams.

The idea of music composition can be broken down and analyzed; the uninitiated might see it as a divine gift whereby inspiration arrives to the composer via some sort of supernatural power channeled through her pencil and forever captured in writing for the world to admire. However, probably all composers have at some point considered and reflected on the notion of systematic composition and craft which is more convenient whenever inspiration fails you. The truth of it is that music composition can be considered an algorithmic process all together, even in the romantic setting presented earlier, however more or less intuitive or conscious. The reader should be reminded of the fact that it is often said that music and math are intimately related, as stated by Ligeti in the quote in the opening of this work.

Also good to reflect on at this early level is the implication of composing by means of a system; allowing a computer to compose autonomously amounts to somehow supplying a system for the computer to compose by, after which the computer forms the composition by carrying out the tasks as specified by the system. There is, however, no intrinsic difference in notion between setting up a system manually and executing it by hand yielding a composition, and programming a computer to somehow do the same. Another philosophically intriguing question is whether there is a difference between *recognizing* and *composing* qualitative music. The connection to algorithmic composition becomes obvious in the context of a random music piece generator which a composer can choose to save pieces from that she recognizes as qualitative. Who is the rightful composer here? The composer or the recognizer ***(Jacob, 1996)***?

3.1 Historical Notes

One might have different opinions on what constitutes the earliest example of algorithmic composition, but some argue that the contributions of music theorist Guido D'Arezzo, inventor of the solmization technique, are most noteworthy. D'Arezzo invented in the 10th century A.D. a technique for automatic conversion from religious texts to music by associating syllables with different (perhaps relative) pitches ***(Nierhaus, 2009)***.

By the time of Palestrina and the renaissance era of music, there were strict rules in place governing what intervals were accepted and how different voices would move, relative to one another, in order for the polyphony of the music to be preserved. Later on, during the baroque era with Bach and Händel, these rules were expanded and more dissonant sounds were allowed which, in turn, required even more rules in terms of how dissonances should be resolved. Both of these traditions are studied in the topic *counterpoint* (intimately associated with the concept of polyphony) which often is divided into three sub-topics out of which Palestrina and Bach counterpoint are the first two. One may consider these systems as rationalized in retrospect and simply motivated by what sounded best, but the fact of the matter is that there are numerous cases where mathematical sequences or ratios have clearly been used as a means of guidance in the composition process. For example in Dufay's *Nuper Rosarum Flores* the ratio 6:4:2:3 is heavily emphasized ***(Edwards, 2011)***.

Also in the Baroque era, around 1650, Athanasius Kircher wrote in his book *Musurgia Universalis* about a mechanical device looking like a wooden box called Arca Musarithmetica ***(Stange-Elbe, 2015)*** which was equipped with levers and knobs regulating different musical aspects of the music that came out of it. The box offered a way to compose with extra-musical means and the combinatorial possibilities of the machine were many.

During the classic era of Mozart and Haydn, the so-called musical dice game (Musikalisches Würfelspiel) was introduced in which a dice was rolled governing how precomposed pieces of music or lyrics would be fitted together ***(Wikipedia, 2018d)***. Other versions with, for example, playing cards existed as well and the game was first seen in 1757. It has been suggested that both Mozart and Haydn made their versions of these games ***(Wikipedia, 2018d)***.

In the early 20th century after World War I, there was presumably a strong wish to break with German music tradition and composers left the romantic sound for a far more dissonant modern era, reflecting the shattered Europe and disturbing times. One of the earliest pioneers was Arnold Schönberg who invented the twelve tone system which formulated strict rules governing which notes the composer may choose from at what time. The basic idea of the system is to get away from traditional tonality and allow all twelve pitch classes to be equally frequent ***(Wikipedia, 2018e)*** resulting in far more dissonant music than what had been seen during the romantic era. Any political insinuations arising from this equality between the twelve pitches classes are left to the reader.

In the 1950's, avantgarde composers continued to investigate different means of composition, sometimes even more from a philosophical viewpoint rather than a musical one. Two of the most famous contributors from this time are John Cage and Karlheinz Stockhausen who both involved randomness in their music, for which the term *aleatoric music* was coined ***(Wikipedia, 2018b)***. At this time, randomness, or chance as it was rather called, would be involved in music in a plethora of ways and it was often in a systematic way that guaranteed a random outcome as far as the compositional idea was concerned. This, in contrast with for example a painter who seemingly randomly throws paint at a canvas but might be governed by some intuitive non-random force ***(Meyer, 1967)***. There is a similar situation in Stockhausen's Klavierstücke written between 1952-1956 ***(Wikipedia, 2018c)***, where one of the pieces consists of fragments and instructions to the player with, among others, the urge to start randomly with a fragment and pick dynamics, pitch and more in the same way ***(Meyer, 1967)***. This might seem as random as the example with the painter and is conversely not random as far as the performer is concerned but in the eyes of the composer, it is indeed random. One can of course argue that music always contains randomness then, since interpretation by a performer always involves a certain portion of originality depending on who is the performer, but this is ultimately a philosophical question. Since composer and performer are often different individuals, randomness has often been introduced in the compositional process this way and everything from graphical scores to exact notation but with (systematic) chance involved has been tried. In one of the most famous pieces by John Cage, *4'33"*, conducted around the middle of the century, the pianist simply reads from the sheets but does not allow the piano to make any sounds. Originally, the windows to the concert hall would be open and so Cage wanted to make a piece that both contained chance, due to different sounds flowing in through the windows every time it was played, as well as convey the philosophical remark that everything could be considered music, not only that which is explicitly broadcast as music ***(Wikipedia, 2018a)***. Needless to say, the piece is often misunderstood. Cage was a Zen Buddhist and describes how he uses the *I Ching*, an old Chinese text, as a composition tool and source of randomness ***(Cage, 1961)***.

Later in the century, technological advancements such as tape recorders and other audio processing devices for home use gave rise to new systematic ways to produce music, one example of which is the minimalist composition *It's gonna rain* by Steve Reich in 1965 where sampled speech is repeated in multiple channels with slight continuous juxtaposition of phase which alters the experience throughout the rather long piece.

3.2 The Computer Era

By the turn of the century, the innovation of computers was well-spread and even though these were not yet available for home use, they could be found in universities lending themselves to be used by scientists and engineers. It is also in the intersection between science and music that we find the earliest uses of computers to generate music.

3.2.1 Algorithmic Composition

Executing a computer program that generates music is a process that has had a few different names. *Computer music* seems to be a very broad term whereas *algorithmic composition* or *computer-generated music* is more specifically the kind of music that is either generated or generated and played by a computer, no matter if this is done with a deterministic or stochastic algorithm. Further expressions that are close to algorithmic composition are *evolutionary music* and *generative music* which both refer to two specific kinds of music that are typically created by computer programs. *Machine improvisation* is a term that usually denotes the use of statistical tools to analyze and further generate new music similar to the learning material. It may be considered a new composition in the same style or an improvisation on the previous material, maybe in part depending on the genre of music. Philosophically however, there is no inherent difference between "true" composition and composition by means of inspiration from other materials (aren't we all inspired by others?) and therefore there is no obvious reason to uphold and further root the use of an expression like machine improvisation.

One objection to even using any of these terms is stated by composer Barlow and he wonders why we should call the "new" music *computer music* when we never called the old music *pencil music* ***(Supper, 2001)***. Alas, the problem of the norm and its impact on deviations from it remains a problem within all realms.

In this thesis, the term algorithmic composition will refer to any system for music composition that is typically executed on a computer or involves the use of algorithms from the scientific field known as information theory (the umbrella under which the field of computer science rests) and that generates music in some representation.

As in general algorithm theory, it is good to distinguish between *deterministic* algorithms and *non-deterministic* ones; deterministic algorithms always follow the exact same recipe and with the same input, they always generate the same output. The same does not apply to non-deterministic algorithms that involve randomness and that may yield different output with the same input on two consecutive executions. One must be reminded, however, that one way to yield different outputs from the same deterministic algorithm is to supply different (random?) inputs which do not make the actual algorithm non-deterministic. With respect to the subject at hand, it is natural that one wishes for an algorithm candidate to yield different pieces of music every time it is executed.

3.2.2 Groupings

There are many thoughts on how to group the different ways that a computer can compose autonomously. In one survey over algorithmic composition the following groups of techniques are listed ***(Fernàndez and Vico, 2013)***:

- Grammars
- Symbolic, Knowledge-Based Systems
- Markov Chains
- Artificial Neural Networks
- Evolutionary and Other Population-Based Methods
- Self-Similarity and Cellular Automata

A similar division had been used before ***(Nierhaus, 2009)***:

- Markov Models
- Generative Grammars
- Transition Networks
- Chaos and Self-similarity
- Genetic Algorithms
- Cellular Automata
- Artificial Neural Networks
- Artificial Intelligence

Needless to say, the same actual results are presented nonetheless. Yet another survey uses a slightly different categorization where is also emphasized the problems with using categories since different methods in different groups can be considered the same ***(Papadopoulos and Wiggins, 1999)***.

All of these categorizations uses the generation method to determine which category a certain method should belong in. This is not necessarily the most important aspect (one might argue that the result is) and when scrutinized, there doesn't seem to exist any clear distinction between some categories. Nonetheless, a similar categorization will be used in this literature survey with neural networks as a modelling strategy being treated separately.

An alternative categorization emphasizing intention of the method used instead of the method itself has also been used ***(Supper, 2001)***:

- Modelling traditional, non-algorithmic compositional procedures
- Modelling new, original compositional procedures, different from those known before
- Selecting algorithms from extra-musical disciplines

Even here, of course, ambiguities exist since an algorithm can both be taken from extra-musical disciplines as well as attempt to model traditional and non-algorithmic compositional procedures.

3.2.3 Contemporary Alternatives Without Neural Networks

Since musical composition is not an exact science, using a computer to compose can be done in any suitable way, and only the creativity of the composer sets the limits. We will now take a look at some general and particular techniques that have been experimented with over the years.

3.2.3.1 The Naive Way

The most straightforward method for algorithmic composition is probably to use books on music theory and set up a framework of rules for how music should be written; how chords should progress, how different voices should relate to each other and much more. Such a program would be elaborate but with a computer, millions of combinations and possible outcomes could be attempted per second and so it would probably yield some good results. Some kind of randomness would have to be involved to yield different results every time the program runs. Also, different rules could be implemented depending on whether we wanted to create music that sounds like baroque music or European schlager from the 20th century. Finally, a good computer representation of music would have to be devised.

One can also choose to add a workflow that is somewhat reminiscent of that used during an old-fashioned compositional process. This was attempted by Jacob where he constructed ideas and materials by first creating motifs, then phrases and then vary these, much like he would have done while composing without a computer ***(Jacob, 1996)***.

A more computer-oriented way of expressing the above is to say that one attempts to solve a *Constraint Satisfaction Problem* (CSP) where we have constraints and try to satisfy as many of them as we can. One such example is the monumental program CHORAL that can write and harmonize Bach chorals with high precision. CHORAL is providing output as results to a 350-constraint CSP ***(Fernàndez and Vico, 2013)***.

3.2.3.2 Mathematics

If the naive way is simply attempting to replicate what has been done, albeit in a new way, one of the most intuitively appealing ways to introduce a systematic process in one's composition is to use mathematics. Of course, mathematics form an important part of some of the methods to be described further down but it can also be applied arbitrarily in whatever system or manner the composer wishes. Both Xenakis and Ligeti are said to have included mathematical processes on different levels in their compositions ***(Edwards, 2011)***.

As one survey groups algorithmic composition methods, there was one method that didn't fit in any group, namely the use of Discrete Fourier Transform (DFT) to create variations on a piece ***(Fernàndez and Vico, 2013)***.

3.2.3.3 Genetic Algorithms

In *genetic algorithms*, a population of solutions are generated in each time step and only the best, according to some scoring function, survive to the next sieving step. This has also been done in algorithmic composition.

One of the most famous examples of genetic algorithms and evolutionary methods used in algorithmic compositions is the project Melomics in which the hardware Iamus manages an ever-growing population of musical themes and fragments. The system allegedly composed the first piece ever to have been written without human intervention on October 15 in 2010 and orchestras continuously play music written by this software ***(Diaz-Jerez, 2011)***.

3.2.3.4 Markov Models or Markov Chains

Andrey Andreyevich Markov (1856-1922) started his work resulting in *Markov models*, used to model an abstract machine with different states that given a time sequence will transition between its states with different probabilities, and the relevant discourse in 1906. The *transition matrix* (probabilities that transition will occur from one state to another) and *initial state* vector are known beforehand and exactly what constitutes a state is variable. In conjunction with music, a state can be considered the subsequent addition of a specific melodic fragment or pitch, this is up to the composer. Markov models, also known as *Markov processes* or *Markov chains*, where the transitions only depend on the last state are called *first order Markov models* but higher orders exist as well (though first order models are most common). The *first order Markov assumption*, one of the fundamental properties of a first order Markov process, is that the transition probabilities only depend on the last seen state, and no states before that. This assumption is not to be taken for granted in all situations. Markov model matrices can either be arrived at by random sampling or by analysis of an existing piece ***(Nierhaus, 2009)***.

Hidden Markov models (HMMs) are Markov models where the transition matrix and initial state matrix are unknown and where so-called *emissions* are the only component of the model possible to inspect. In such a setting, the emissions are considered the musical property that one wants to generate and by using some well-known algorithms (*forward-backward algorithm*, *Viterbi algorithm* and *Baum-Welch algorithm*) one can after every single inspected emission determine a most likely underlying hidden Markov model to explain the observations.

Both hidden and regular Markov models have been used to generate music. Markov models in general were first used in connection to music by Harry F. Olsen and Henry Belar inventing the world's first synthesizer ***(Nierhaus, 2009)***. In the Illiac Suite, the world's first composition by a computer named after the Illiac computer in Chicago, both generative grammars and Markov models were used ***(Nierhaus, 2009)***. The suite is divided in four pieces called experiments, and in the fourth experiment, the conductors of the experiments, Lejaren Hiller and Leonard Isaacson describes in how they looked in to Markov processes of orders zero, one and even higher ***(Hiller and Isaacson, 1959)***. Later on, Iannis Xenakis, being both an engineer and a composer, used Markov models in his composition as well. For example in *Analogique A.*, first order Markov chains are used to determine the order of segments ***(Nierhaus, 2009)***. Xenakis has also himself given insight into how he uses Markov processes ***(Xenakis, 1992)***. The first HMM used for algorithmic composition was deployed in 2001 by Farbood and Schoner ***(Fernàndez and Vico, 2013)***.

Using Markov models as a tool for algorithmic composition has been well experimented with ever since the days of Lejaren, Hiller and Xenakis and today, they form the basis for music generation programs *M* and *Jam Factory* ***(Jacob, 1996)***. One stated opinion, however, is that Markov chains do not account for a good alternative for algorithmic composition and that low order models gave rise to "strange, unmusical compositions that wandered aimlessly". High order models on the other hand are computationally expensive and more or less mimics the music on which the Markov model was trained ***(Fernàndez and Vico, 2013)***.

Much later, a variable length Markov model called *prediction suffix tree* (PST) was invented and has also been used for algorithmic composition ***(Ron et al., 1996)***.

3.2.3.5 Grammars and L-Systems

Grammars are used to define the hierarchical rules of some form of language and are therefore also well suited to define music ***(Nierhaus, 2009)***. There is a large number of different types of grammars, some involving randomness, non-determinacy and recursion where some symbols may lead to entirely new grammars, suitable to model hierarchical phenomenons. Since grammars can be used to define a language, they may also generate strings of the language by simply randomizing or by other means determining a set of symbols occurring in the grammar. These may then be successively replaced as suitable according to the rules of the grammar. One idea is that music follows patterns, often both on a large and small scale simultaneously and, intuitively, by using a grammar with a logic derivation order in it, these patterns have a good chance to occur. In the previously mentioned Illiac Suite written in 1956, grammars are used to generate a part of the music ***(Baggi, 1998)***.

One particular kind of grammar is the L-System described by Lindenmayer originally meant to model the structure and growth of biological entities ***(Lindenmayer, 1968a, Lindenmayer, 1968b)***. L-Systems exist in different types and differs from regular grammars in two ways. First of all, there are no terminals which means that a token can be expanded forever resulting in an infinite number of combinations of symbols ***(Nierhaus, 2009)***. Also, in an L-System, expansion takes place in parallell and multiple symbols are expanded at once as opposed to sequential expansion. L-Systems later got graphical interpretations and can give rise to trees and plants that look extremely realistic. In 1986, Prusinkiewicz extended the use of L-Systems and showed how they can be used to generate music ***(Prusinkiewicz, 1986)***. The method is rather arbitrary and in the same way as L-Systems can be interpreted graphically, Prusinkiewicz shows that by assigning musical meaning to symbols, music can be generated. Several L-Systems can be connected to account for multiple dimensions of music such as chords, dynamics, rhythms and more and by using other versions of L-Systems, randomness can be introduced in different ways. One of the first to use L-Systems in musical composition was the composer Kyburz in the piece *Cells* written in 1993-1994 ***(Supper, 2001)***.

Worth to mention is also Experiments in Musical Intelligence (EMI) which is a system presented by David Cope starting in 1981 ***(Nierhaus, 2009)***. Cope related his work to a musical Turing test and thought about the compositional process as changing existing material by means of rearranging order and reconstruct smaller building blocks in a similar manner. The work was done using transition networks which in its essence is a graphical automata-based representation of a grammar where a network consists of many subnetwork which each corresponds to treatment of a non-terminal or terminal grammar symbol. Much like today's neural networks, input training data was used and broken down into small fractions from which systematic reconstruction using similar ideas and properties could take place along with rearrangement of these pieces.

Another system presented in 2015 is based on principles of splicing ***(De Felice et al., 2015)***. Splicing occurs biologically in DNA whereby two strains are cut and form two new strings by switching parts with each other. A splicing system consists of an alphabet, an initial set of words and a set of splicing rules which can act recursively on the initial set of words, adding new words to the vocabulary continuously. The authors use 4-part chorales and chords as words and combine them using splicing to form new words. After some time, they choose the word, corresponding to a composed choral, by using a fitness value for generated words. The same authors later refined their system by adding more complex musical information to the splicing rules ***(De Felice et al., 2017)***.

Under this paragraph, one may also mention *fractals* which are geometric shapes that are found in nature and that show structure and repetition on both macro and micro levels. They have a mathematical foundation and may be and have been used for musical composition purposes.

3.2.3.6 AI Algorithms

Even though the term artificial intelligence (AI) is somewhat ambiguous and many of the different progressions in algorithmic compositions that have been made in computers in the past may fall under this category, the classic AI algorithms involves, but are not restricted to, the search for solutions in for example combinatoric board games or path finding when the solution space is huge. Often both heuristics and backtracking is employed to narrow down the set of solutions and provide some systematic way of searching all solutions within some smaller subspace. For example, Gill used in 1963 an AI approach to compose twelve-tone melodies where backtracking was used to satisfy a set of predetermined rules ***(Fernàndez and Vico, 2013)***.

3.2.4 Modelling Music with Neural Networks

Artificial neural networks were developed around the middle of the 20th century and were first used in connection with music in the 70's and 80's to classify music ***(Nierhaus, 2009)***. At the end of the 1980's, the connectionist paradigm, as the approach to use neural networks was called at the time, began to get attention as a promising choice for modelling cognitive processes. Lischka describes in 1987 an abstract cognitive model for creative musical tasks and he confirms that the new connectionist approach is superior to earlier symbolic (grammar) approaches ***(Lischka, 1987, Lischka, 1991)***. A simple example with a Boltzmann machine classifying harmonic functions based on present pitches is showed and Lischka argues that the neural networks, being inspired by a real cognitive system (neurons and the brain) can manage to learn rules that are less specific and concrete than what is necessary for a symbolic system to perform well.

The era of neural networks and music modelling can roughly be divided into three periods. In the first part (1987-2001) (here referred to as the *early* era or period), SRNs were largely used with hand-crafted data and very small training sets. Listening samples are missing and evaluation is often neglected. During this period, much is focused on guiding the networks with respect to both data representation, conditioning on extra inputs and network connectivity (fixed weights and partial connectivity for example). The second part (2002-2011) (here referred to as the *middle* era or period) begins with Eck and Schmidhuber using the, by then, new LSTM architecture for blues music ***(Eck and Schmidhuber, 2002)***. This is a middle era during which the equations of the RNNs and LSTMs had been completely established and subsequently, these architectures are heavily used whereas we, consequently, see less of SRNs. Some researchers start to investigate other architectures as well and the tendency is to go towards a more formalized training and evaluation procedure. Listening samples online are now becoming more common. The third period (2012-now) (here referred to as the *recent* era or period) is a time where all sorts of architectures are tried. This period begins with Boulanger-Lewandowski et al. who set a baseline on a number of datasets which they also release preprocessed versions of ***(Boulanger-Lewandowski et al. (RNN-RBM, RNN-NADE), 2012)***. As a result, music modelling becomes a benchmark and the trend to try out all sorts of improvements on recurrent models, using the baselines (and often also the mentioned preprocessed datasets) of this publication as a reference, gains in popularity. Researchers dealing with music as a benchmark only often test their models in several different fields (language is also a popular choice) in parallel and as far as music is concerned, only measure the test set results for comparison with previous baseline. Thus, the compositional capabilities of these models are not tried out to any further extent and in the section to come, these models are not referenced in all sections in the same way as models that purposely aim to model music. This is also because these publications use input and data representations that are copied from the baseline publications which they are trying to beat, and thus, contribute with no novelties in these areas. Subsequently, the mentioned datasets fast becomes a new standard in an era where large training sets, automated preprocessing and listening samples available online is the standard. During this period, the paradigm is that networks are trained end to end with music, as is, thus with a minimum amount of guidance.

This section is intended to give a fairly detailed overview over how neural networks have been used for generative musical tasks. Because inspiration and valuable insights into how data is processed and music is modelled may be acquired from other different problem contexts involving music; these tasks are not necessarily of purely compositional nature. Because of the large body of treated material, this section is divided into sections treating different, fairly specific, aspects of task procedures to give the reader a better and more insightful overview. The chosen division is by no means the only suitable and in the currently only survey on algorithmic composition with a large focus on neural networks, the following perspectives are used instead ***(Briot et al., 2017)***:

- **Objective**
- **Representation**
- **Architecture**
- **Strategy**

In this literature survey, all processed material has been considered with twelve aspects in mind:

- **Application / Purpose**: what is the *primary* purpose of the model?
- **Domain**: what representation of music is being modelled?
- **Musical domain**: what are the limitations of the music being modelled?
- **Genre**: what musical genre is modelled?
- **Input representation**: what is the representation of the music before it enters the preprocessing?
- **Data representation**: what is the representation of the music internally as it is processed by the network (e.g. after preprocessing)?

- **Model / architecture**: what does the layout of the model look like and what sets it apart from other models?
- **Frameworks**: what preprocessing or machine learning framework have been used, if any?
- **Datasets**: what datasets have been used?
- **Evaluation**: how is the model evaluated?
- **Source code**: is the source code available and if so, where?
- **Samples**: are there generated or recorded samples available and if so, where?

Where convenient, these facets have been further divided into exhaustive sets of categories to further give an efficient demonstration of the variety.

As a final remark, one might think of traditional and deterministic algorithms as tools which we have to encode ourselves whereas with neural networks, depending on how we model things, there is a chance that the network can discover latent factors that we are unaware of. Thus, one might say that algorithmic composition potentially takes a step from being a tool for making music towards becoming a composer, capable of discovering latent aesthetic patterns and acting more independently of the programmer, with the introduction of neural networks.

3.2.4.1 Application / Purpose

Music has been modeled with neural networks for other reasons than composition, even though this purpose has been one of the most common. In this section, the high-level function that we wish for a model in question to fulfill, and which is subsequently evaluated, is discussed. Some models can perform several tasks depending on the mode of operation and some models could potentially perform several tasks but are only used for one. In the latter case, the model will, as previously indicated, only be listed under the subcategory that it was actually evaluated with respect to.

- **Composition:**

  The large corpus of material regarding music modelling with neural networks describes models that should be able to compose melodies ***(Todd, 1988; Todd, 1989; Mozer (CONCERT), 1990; Lewis (CBR), 1991; Mozer and Soukup (CONCERT), 1991; Tsang and Bellgard (EBM), 1992; Bellgard and Tsang (EBM), 1994; Nishijima and Watanabe (Neuro-Musician), 1993; Chen and Miikkulainen, 2001; Franklin (CHIME), 2001; Franklin, 2004; Franklin and Locke, 2005; Verbeurgt et al., 2004; Franklin, 2006; Franklin, 2005; Corrêa et al., 2008; Bickerman et al. (RBM-provisor), 2010; Coca et al., 2011; Bretan et al., 2016; Colombo et al., 2016; Jaques et al. (RL Tuner), 2016; Sturm et al. (char-rnn, folk-rnn), 2016; Sun et al., 2016; Agarwala et al., 2017; Chen et al. (FusionGAN),***

2017; Colombo et al. (DAC), 2017; Guimaraes et al. (ORGAN), 2017; Hadjeres et al. (GLSR-VAE), 2017; Hadjeres and Nielsen (Anticipation-RNN), 2017; Jaques et al. (Sequence Tutor), 2017; Roberts et al. (MusicVAE), 2017; Roberts et al. (MusicVAE), 2018; Tikhonov and Yamshchikov (VRASH), 2017; Wu et al. (HRNN), 2017; Eppe et al., 2018; Walder and Kim (MotifNet), 2018), a progression of harmonies ***(Lewis (CBR), 1991; Tsang and Bellgard (EBM), 1992; Bellgard and Tsang (EBM), 1994; Gang and Berger, 1996; Berger and Gang, 1997; Berger and Gang, 1999; Melo and Wiggins, 2003; Franklin, 2004; Franklin and Locke, 2005; Choi et al. (char-RNN, word-RNN), 2016)*** or full-stack polyphonic music ***(Melo and Wiggins, 2003; Boulanger-Lewandowski et al. (RNN-RBM, RNN-NADE), 2012; Goel et al. (RNN-DBN), 2014; Liu and Ramakrishnan, 2014; Fabius and van Amersfoort (VRAE), 2015; Gan et al. (TSBN), 2015; Lyu et al. (LSTM-RTRBM), 2015; Sun (DeepHear), 2015; Vohra et al. (LSTM-DBN), 2015; Huang and Wu, 2016; Lattner et al. (C-RBM), 2016; Mogren (C-RNN-GAN), 2016; O'Brien and Román (MusicNet), 2016; Walder, 2016; Brunner et al. (JamBot), 2017; Johnson (LSTM-NADE, TP-LSTM-NADE, BALSTM), 2017; Dong et al. (MuseGAN), 2017; Hennig et al. (Classifying VAE, Classifying VAE+LSTM), 2017; Sabathé et al. (DRAW), 2017; Simon and Oore (PerformanceRNN), 2017; Ycart and Benetos, 2017; Colombo and Gerstner (BachProp), 2018; Koh et al. (C-RVAE), 2018; Mao et al. (DeepJ), 2018; Simon et al. (MusicVAE), 2018)*** sometimes only with a restricted number of voices ***(Freisleben, 1992; Goldman et al. (NetNeg), 1999; Hadjeres and Pachet (DeepBach), 2017; Huang et al. (CocoNet), 2017; Liang et al. (BachBot), 2017; Roberts et al. (MusicVAE), 2017; Roberts et al. (MusicVAE), 2018)***. Some models also provide melody along with a chordal accompaniment ***(Mozer (CONCERT), 1994; Tsang and Bellgard (EBM), 1992; Bellgard and Tsang (EBM), 1994; Eck and Schmidhuber, 2002; Eck and LaPalme, 2006; De Prisco et al., 2017; Lee et al. (SeqGAN), 2017; Shin et al., 2017; Teng et al., 2017; Yang et al. (MidiNet), 2017)*** and sometimes with percussion as well ***(Chu et al., 2016)***. Composition has also been done in the audio domain ***(van den Oord et al. (WaveNet), 2016a; Mehri et al. (SampleRNN), 2017)***, or a representation thereof ***(Sarroff and Casey (DeepAuto-Controller), 2014; Nayebi and Vitelli (GRUV), 2015; Kalingeri and Grandhe, 2016)***.

- **Classification:**

  Even though most often, as a sub-task, related to the modelling of composition, pure classification has also been done which involves solving the problem of modelling music. One example is chord classification ***(Laden and Keefe, 1989)***. Typically however, this task is often more connected to Music Information Retrieval (MIR) which is a related, but different, area than that of general machine learning art and more specifically musical composition.

- **Harmonization:**

  Harmonization is the process of supplying an existing voice with an additional voice (or voices). Less formally, this often implies supplying a melody with a chordal accompaniment where each chord appears for a fixed or variable number of time steps. In essence, this is a pattern completion task (can also be seen as classification) and do not amount to full generation of music on its own. Supplying a melody with accompaniment has been a common task in the history of neural networks ***(Gang and Lehmann, 1995; Gang et al. (HNN), 1997; Gang et al. (HNN), 1999)***, especially Bach chorale harmonization ***(Hild et al. (HARMONET), 1991; Liang et al. (BachBot), 2017)*** or reharmonization ***(Hadjeres and Pachet (DeepBach), 2017)*** but other chorales as well ***(Tsang and Bellgard (EBM), 1992; Bellgard and Tsang (EBM), 1994)***. Creating counterpoint by supplying a melody with a counter-melody according to the rules of counterpoint has also been done ***(Adiloglu and Alpaslan (NeuroComposer), 2007)***. Harmonization can sometimes also involve adding ornaments to the different voices making up the harmonized work ***(Hild et al. (HARMONET), 1991)***.

- **Variation:**

  Variation is the act of taking a piece of music and elaborating it, or ornamenting it, in some sense with the purpose to create something very reminiscent of the original but still new. Examples are ornamentation of quarter notes by occasional insertion of smaller note values in between ***(Hild et al. (HARMONET), 1991)*** or systematic elaboration of quarter notes to sixteenth notes in groups of four ***(Hörnel and Menzel (MELONET), 1998)***. Another group of systems that accomplish variations, even though they can be seen as compositions as well, are systems that are designed to trade jazz solos and produce tunes reminiscent of those produced by the opposite player ***(Nishijima and Watanabe (NeuroMusician), 1993; Franklin (CHIME), 2001)***. Reconstructing harmonic progressions, both by replacing single chords but also reworking entire contexts has also been done ***(Huang et al. (ChordRipple), 2016)***. A similar work lets the user, given a chord, predict chordal context ***(Madjiheurem et al. (Chord2Vec), 2016)***. Finally, composing music adhering strictly to a certain style can be seen as variation in some cases depending on the procedure ***(Koh et al. (C-RVAE), 2018)***.

- **Interpretation:**

  Interpreting music is the action when a musician performs music and bridges the gap between the musical instruction (notated, symbolic sheet music) and the sounding music. In terms of machine learning, this amounts to modelling additional properties, aside from actual pitches and durations, that make music what it is. These properties can, in fact, be written down in the score (such as dynamics, tempo, accents) but not necessarily (micro-timing, vibrato, deviations from written instructions). It has not been a common task to encode interpretation of music only as a main goal but examples exists. One such example models dynamics only

(Malik and Ek (StyleNet), 2017). However, some model these properties with an equally large focus on composition as well ***(Franklin, 2006; Nishijima and Watanabe (Neuro-Musician), 1993; Simon and Oore (PerformanceRNN), 2017)***.

- **Transcription:**

  Automatic Music Transcription (AMT) is the procedure of turning a music audio signal into written music and it is big field on its own. However, a music model may be incorporated into such a system, to improve its performance. Typically, a trained music model may serve as a prior through which the initial transcription is filtered ***(Boulanger-Lewandowski et al. (RNN-RBM, RNN-NADE), 2012; Sigtia et al., 2014; Ycart and Benetos, 2017)***, alternatively, the transcription is retranscribed using the model as a prior ***(Sigtia et al., 2014)***. Such a model is possibly trained in the same way as a model designed to compose but is used for transcription and is measured thereafter.

- **Interactive systems:**

  Some systems are specifically developed with the idea that a user should be able to guide some process, thus working in tandem with the trained model, to accomplish something ***(Hadjeres and Pachet (DeepBach), 2017; Hadjeres and Nielsen (Anticipation-RNN), 2017)***. This is a fairly reasonable idea and even though most systems are designed to work on their own after training, one must ask oneself whether it is fair to expect a network to fill such a role?

  Systems can also be interactive in other ways. For example, Neuro-Musician and CHIME are designed to be deployed in so-called *adlib* sessions where jazz soloists perform a number of measures of solo after which another player is supposed to answer, in a call-and-response fashion, by continuing the solo in a similar style ***(Nishijima and Watanabe (Neuro-Musician), 1993; Franklin (CHIME), 2001)***.

  In some cases, authors build some kind of additional software for users to try out, and through which a model of some kind can be manipulated and explored ***(Sarroff and Casey (DeepAutoController), 2014; Huang et al. (ChordRipple), 2016; Engel et al., 2017; Hadjeres and Pachet (DeepBach), 2017; Mao et al. (DeepJ), 2018)***.

- **Other:**

  Other purposes than the above have also occurred, for example testing whether a reductionist approach to music is trustworthy by compressing and decompressing symbolic music ***(Large et al., 1995; Bretan et al., 2017)***. Sometimes, the neural network part is just a subcomponent of a larger system where the remaining system is not based on neural networks ***(Verbeurgt et al., 2004; De Prisco et al., 2017)***. Other publications show new techniques improving existing ones by reconstructing timbre in the domain of real audio in an autoencoder fashion ***(Engel et al., 2017)***.

3.2.4.2 Domain

What is music, the notes written on the sheet by a composer or the sounding result reaching our ears and filling us with an experience? This section treats what a network, on a higher level than data representation, is trying to model. In most cases, architectures try to model music symbolically, namely as an instruction of how to play music. However, attempts to model music as it sounds have also been done, typically in the domain of audio signals, or transformations thereof. Depending on which, the resulting learning problems might be very different and requirements of specific kind might be placed on the model.

- **Symbolic:**

  In the symbolic domain, written music is modelled in some way. The vast majority of all publications on algorithmic composition with neural networks fall in this category, almost always resulting in a learning problem with ordered discrete classes corresponding to notes or chords, much like the learning problems that arise within NLP or image classification.

- **Signal:**

  Modelling the sounding result of music is still relatively new, but has been done in a few cases. Typically, audio is a stream of events recorded in one or several channels with some sample rate where each sample corresponds to some data. A common sample rate is 44100 samples per second (44.1 kHz) but lower rates, for example 16 kHz, are common too. Each sample is typically a 32-bit or 16-bit integer and it is common to merge all channels to a mono track before modelling.

  With a 16 kHz sample rate and 16-bit samples, speech and also music have been modelled in WaveNet and SampleRNN directly using its wave form ***(van den Oord et al. (WaveNet), 2016a; Mehri et al. (SampleRNN), 2017; Engel et al., 2017)***. Worth to point out is that PCM encoding of 16-bit integers to 8-bit integers were used in these models.

  In other cases, audio has instead been transformed using Fourier analysis into spectrograms which have been modelled instead of the pure audio signal ***(Sarroff and Casey (DeepAutoController), 2014; Nayebi and Vitelli (GRUV), 2015; Kalingeri and Grandhe, 2016)***. Also Mel spectra have used for modelling ***(Eppe et al., 2018)***.

  The sounding result of music can also be modelled without using audio signals by taking into account exactly the aspects of music that an interpreter adds to the experience when playing, which becomes obvious when inspecting the data representation chosen ***(Franklin, 2006; Nishijima and Watanabe (Neuro-Musician), 1993; Simon and Oore (PerformanceRNN), 2017)***.

3.2.4.3 Musical Domain

What aspect of music is being modelled? Music in its entirety, or just music of some structural type? This parameter is quite diverse historically even though three very common goals are to model a melody (monophonic music), harmonies or with no restriction (polyphonic music). Sometimes, music with a melody and chordal background (homophonic music) is the choice of domain. Note that this section refers to the main musical domain that a model should be able to output, not what kind of data is being used to accomplish this. For example, some models use harmonies as conditioning for a melody but the model does not output any chords. Such a model is listed in the monophonic subsection, even though the way the model represents the chords in the conditioning might be listed under the relevant subsection in the section about data representation.

- **Melody (Monophony):**

  Monophonic music, or melodies, have been modelled in all times, perhaps more frequently so in the very early era where a lot of examples exist ***(Todd, 1988; Todd, 1989; Mozer (CONCERT), 1990; Lewis (CBR), 1991; Mozer and Soukup (CONCERT), 1991; Nishijima and Watanabe (Neuro-Musician), 1993; Mozer (CONCERT), 1994; Large et al., 1995; Chen and Miikkulainen, 2001; Franklin (CHIME), 2001)***. Some models work with several voices in a completion mode of operation, which enables them to provide a melody given a harmonic sequence ***(Tsang and Bellgard (EBM), 1992; Bellgard and Tsang (EBM), 1994; Hörnel and Menzel (MELONET), 1998; Bickerman et al. (RBM-provisor), 2010)***. In the middle period, there are also examples of this musical domain ***(Franklin, 2004; Franklin and Locke, 2005; Verbeurgt et al., 2004; Franklin, 2006; Franklin, 2005; Adiloglu and Alpaslan (NeuroComposer), 2007; Corrêa et al., 2008; Coca et al., 2011)***. In the recent era, the modelling of monophony still persists ***(Bretan et al., 2016; Colombo et al., 2016; Jaques et al. (RL Tuner), 2016; Sturm et al. (char-rnn, folk-rnn), 2016; Sun et al., 2016; Agarwala et al., 2017; Chen et al. (FusionGAN), 2017; Colombo et al. (DAC), 2017; Engel et al., 2017; Guimaraes et al. (ORGAN), 2017; Hadjeres et al. (GLSR-VAE), 2017; Hadjeres and Nielsen (Anticipation-RNN), 2017; Jaques et al. (Sequence Tutor), 2017; Roberts et al. (MusicVAE), 2017; Roberts et al. (MusicVAE), 2018; Tikhonov and Yamshchikov (VRASH), 2017; Wu et al. (HRNN), 2017; Walder and Kim (MotifNet), 2018)*** even though less common now. Modelling monophonic music directly from the audio signal has been done as well ***(Eppe et al., 2018)***.

- **Harmony:**

  Harmonies, e. g. sequences of chords, have been modelled, specifically or as a particular subcomponent of a larger model, in the symbolic domain ***(Laden and Keefe, 1989; Hild et al. (HARMONET), 1991; Lewis (CBR), 1991; Tsang and Bellgard (EBM), 1992; Bellgard and Tsang (EBM), 1994; Mozer (CONCERT), 1994; Gang and***

Lehmann, 1995; Gang and Berger, 1996; Berger and Gang, 1997; Berger and Gang, 1999; Hörnel and Menzel (MELONET), 1998; Gang et al. (HNN), 1997; Gang et al. (HNN), 1999; Melo and Wiggins, 2003; Franklin, 2004; Franklin, 2005;Franklin and Locke, 2005; Franklin, 2006; Huang et al. (ChordRipple), 2016; Madjiheurem et al. (Chord2Vec), 2016), sometimes framed as a textual problem ***(Choi et al. (char-RNN, word-RNN), 2016)***.

- **Melody with chords (Homophony):**

  Music consisting of an accompaniment that, roughly, progresses evenly according to some note duration, with a highest voice, melody, that either follows or do not follow the rhythm of the accompaniment is considered in this section.

  One early example is one of the first uses of RBMs in music modelling where chorales were modelled ***(Tsang and Bellgard (EBM), 1992; Bellgard and Tsang (EBM), 1994)***. In the last version of CONCERT, the model has been extended to use chords as well and can now model melody and chords as well as melody only, as previously ***(Mozer (CONCERT), 1994)***. Some examples focus on melody and chords but with a model that could potentially model polyphony as well ***(Eck and Schmidhuber, 2002)***. One common setup models melody and chords with different representations but simultaneously ***(Eck and LaPalme, 2006; De Prisco et al., 2017; Lee et al. (SeqGAN), 2017; Shin et al., 2017)***. In some cases, rhythm is added as well ***(Chu et al., 2016)***. Melody may also be generated conditioned on a previously generated chord sequence ***(Teng et al., 2017; Yang et al. (MidiNet), 2017)***.

- **Polyphony:**

  Polyphony here amounts to free movement in more than one voice simultaneously, contrasted with multiple voices but where all except one belongs to a chord, symbolically or as several notes. Polyphony is often characterized by counterpoint, which is the musical technique where no voice, in theory, is more important than any other, thus they all constitute qualitative melodies. This is in contrast with homophony where the division in lead (or melody) and accompaniment is apparent. In practice, however, in polyphonic music, the highest voice often appears as the melody to the listener.

  Modelling polyphonic music was done already in the early era modelling two-voice music ***(Freisleben, 1992; Goldman et al. (NetNeg), 1999)***. In the middle era, data representations and output procedures that made polyphonic music possible became more common which was often taken advantage of ***(Melo and Wiggins, 2003)***, however not always ***(Eck and Schmidhuber, 2002)***. The boom of modelling polyphonic music with neural networks belongs to the recent era ***(Boulanger-Lewandowski et al. (RNN-RBM, RNN-NADE), 2012; Goel et al. (RNN-DBN), 2014; Liu and Ramakrishnan, 2014; Sigtia et al., 2014; Fabius and van Amersfoort (VRAE), 2015;***

Gan et al. (TSBN), 2015; Lyu et al. (LSTM-RTRBM), 2015; Sun (DeepHear), 2015; Vohra et al. (LSTM-DBN), 2015; Huang and Wu, 2016; Lattner et al. (C-RBM), 2016; Mogren (C-RNN-GAN), 2016; O'Brien and Román (MusicNet), 2016; Walder, 2016; Bretan et al., 2017; Brunner et al. (JamBot), 2017; Johnson (LSTM-NADE, TP-LSTM-NADE, BALSTM), 2017; Dong et al. (MuseGAN), 2017; Hadjeres and Pachet (DeepBach), 2017; Hennig et al. (Classifying VAE, Classifying VAE+LSTM), 2017; Huang et al. (CocoNet), 2017; Liang et al. (BachBot), 2017; Malik and Ek (StyleNet), 2017; Roberts et al. (MusicVAE), 2017; Roberts et al. (MusicVAE), 2018; Sabathé et al. (DRAW), 2017; Simon and Oore (PerformanceRNN), 2017; Ycart and Benetos, 2017; Colombo and Gerstner (BachProp), 2018; Koh et al. (C-RVAE), 2018; Mao et al. (DeepJ), 2018; Simon et al. (MusicVAE), 2018) where it is the standard rather than something ambitious.

In the case of modelling music straight from an audio signal, or a representation thereof, ***(Sarroff and Casey (DeepAutoController), 2014; Nayebi and Vitelli (GRUV), 2015; Kalingeri and Grandhe, 2016; van den Oord et al. (WaveNet), 2016a; Mehri et al. (SampleRNN), 2017)***, there is no restriction in the musical domain, and so, if the music is polyphonic, these models fall under here as well.

- **Percussion:**

  Attempts to model rhythmic instruments have also been made, in isolation ***(Choi et al. (char-RNN, word-RNN), 2016)*** or as part of a larger system ***(Chu et al., 2016; Roberts et al. (MusicVAE), 2017; Roberts et al. (MusicVAE), 2018)***, and can be seen as both polyphony or monophony, depending on what kind of rhythm is modelled.

3.2.4.4 Genre

A lot of publications do not reveal the exact genre of what is trained on, but in hindsight, most genres are considered at some point. In general, publications choose to only use music in common time or alla breve (subdivision of the measure in four or two) ***(Eck and LaPalme, 2006; Sun et al., 2016; Bretan et al., 2017; Johnson (LSTM-NADE, TP-LSTM-NADE, BALSTM), 2017; Malik and Ek (StyleNet), 2017; Roberts et al. (MusicVAE), 2017; Roberts et al. (MusicVAE), 2018; Teng et al., 2017; Wu et al. (HRNN), 2017)*** even though counter-examples to this exist as well ***(Mozer (CONCERT), 1990; Mozer and Soukup (CONCERT), 1991; Boulanger-Lewandowski et al. (RNN-RBM, RNN-NADE), 2012; Lyu et al. (LSTM-RTRBM), 2015; Simon et al. (MusicVAE), 2018)***.

The classical genre in general ***(Boulanger-Lewandowski et al. (RNN-RBM, RNN-NADE), 2012; Goel et al. (RNN-DBN), 2014; Gan et al. (TSBN), 2015; Lyu et al. (LSTM-RTRBM), 2015; Vohra et al. (LSTM-DBN), 2015; Huang and Wu, 2016; Madjiheurem et al. (Chord2Vec), 2016; Mogren (C-RNN-GAN), 2016; Walder, 2016;***

Bretan et al., 2017; Johnson (LSTM-NADE, TP-LSTM-NADE, BALSTM), 2017; Hennig et al. (Classifying VAE, Classifying VAE+LSTM), 2017; Malik and Ek (StyleNet), 2017; Simon and Oore (PerformanceRNN), 2017; Wu et al. (HRNN), 2017; Ycart and Benetos, 2017; Mao et al. (DeepJ), 2018; Walder and Kim (MotifNet), 2018) and especially chorales have been very popular due to the straightforwardness and relative complexity. Bach has been a popular choice ***(Duff, 1989; Mozer (CONCERT), 1990; Hild et al. (HARMONET), 1991; Mozer and Soukup (CONCERT), 1991; Mozer (CONCERT), 1994; Verbeurgt et al., 2004; Boulanger-Lewandowski et al. (RNN-RBM, RNN-NADE), 2012; Goel et al. (RNN-DBN), 2014; Liu and Ramakrishnan, 2014; Sigtia et al., 2014; Gan et al. (TSBN), 2015; Lyu et al. (LSTM-RTRBM), 2015; Vohra et al. (LSTM-DBN), 2015; Huang et al. (ChordRipple), 2016; Huang and Wu, 2016; Madjiheurem et al. (Chord2Vec), 2016; O'Brien and Román (MusicNet), 2016; Walder, 2016; Johnson (LSTM-NADE, TP-LSTM-NADE, BALSTM), 2017; Hadjeres et al. (GLSR-VAE), 2017; Hadjeres and Pachet (DeepBach), 2017; Hadjeres and Nielsen (Anticipation-RNN), 2017; Hennig et al. (Classifying VAE, Classifying VAE+LSTM), 2017; Huang et al. (CocoNet), 2017; Liang et al. (BachBot), 2017; Colombo and Gerstner (BachProp), 2018; Walder and Kim (MotifNet), 2018)*** as well as Vivaldi ***(Coca et al., 2011)***, Mozart ***(Freisleben, 1992; Gang and Berger, 1996; Lattner et al. (C-RBM), 2016; Colombo and Gerstner (BachProp), 2018)***, Haydn ***(Gang and Berger, 1996; Berger and Gang, 1999; Colombo and Gerstner (BachProp), 2018)***, Bartók ***(Chen and Miikkulainen, 2001)***, Pachelbel ***(Hörnel and Menzel (MELONET), 1998)***, Prokofiev ***(Melo and Wiggins, 2003)*** and Beethoven ***(Mehri et al. (SampleRNN), 2017; Sabathé et al. (DRAW), 2017)***. Counterpoint, as used during the renaissance, has also been modelled ***(Goldman et al. (NetNeg), 1999; Adiloglu and Alpaslan (NeuroComposer), 2007)***.

Jazz has been modelled as well ***(Franklin, 2004; Franklin and Locke, 2005; Franklin, 2006; Franklin, 2005; Bickerman et al. (RBM-provisor), 2010; Choi et al. (char-RNN, word-RNN), 2016; Chen et al. (FusionGAN), 2017; De Prisco et al., 2017; Malik and Ek (StyleNet), 2017)*** which often involves modelling other aspects of the music, such as dynamics and timing, which are extra important in the jazz genre. This has been done either in isolation or simply as an extra ingredient in plain modelling of music ***(Nishijima and Watanabe (Neuro-Musician), 1993; Franklin (CHIME), 2001)***.

Often for melody and chords but for other categories as well, different books or datasets with standard pop, rock and jazz songs have been sources for material ***(Mozer (CONCERT), 1994; Bickerman et al. (RBM-provisor), 2010; Bretan et al., 2016; Chu et al., 2016; Huang et al. (ChordRipple), 2016; Shin et al., 2017; Teng et al., 2017; Wu et al. (HRNN), 2017; Yang et al. (MidiNet), 2017)***, especially in the early era when it was common with small training sets and hand-crafted data.

Blues progressions have been used as training data, appearing as such for the first time in the middle period ***(Eck and Schmidhuber, 2002)***.

Folk music became popular from the middle era and on ***(Boulanger-Lewandowski et al. (RNN-RBM, RNN-NADE), 2012; Goel et al. (RNN-DBN), 2014; Sigtia et al., 2014; Gan et al. (TSBN), 2015; Vohra et al. (LSTM-DBN), 2015; Madjiheurem et al. (Chord2Vec), 2016; Walder, 2016; Chen et al. (FusionGAN), 2017; Johnson (LSTM-NADE, TP-LSTM-NADE, BALSTM), 2017; Guimaraes et al. (ORGAN), 2017; Lee et al. (SeqGAN), 2017; Wu et al. (HRNN), 2017; Colombo and Gerstner (Bach-Prop), 2018; Walder and Kim (MotifNet), 2018)*** and Irish folk has been used repeatedly ***(Eck and LaPalme, 2006; Colombo et al., 2016; Sturm et al. (char-rnn, folk-rnn), 2016; Agarwala et al., 2017; Colombo et al. (DAC), 2017)***. Other examples uses Brazilian folk music ***(Corrêa et al., 2008)***. Other traditional music, such as Klezmer ***(Colombo et al. (DAC), 2017)***, has been attempted too.

Some authors like to model their favourite artist, for example David Bowie ***(Nayebi and Vitelli (GRUV), 2015)***, Metallica ***(Choi et al. (char-RNN, word-RNN), 2016)*** or favourite genre, such as ragtime ***(Sun (DeepHear), 2015)*** or just piano music, either from different artists ***(Kalingeri and Grandhe, 2016; Sun et al., 2016; van den Oord et al. (WaveNet), 2016a)*** or specific ***(Lattner et al. (C-RBM), 2016; Bretan et al., 2017)***.

Modelling video game music has been done as well ***(Fabius and van Amersfoort (VRAE), 2015; Chu et al., 2016; Koh et al. (C-RVAE), 2018)*** and work in this genre may be of special importance as it is sometimes seen as desirable to automate composition of video game music so that it correlates with the current mood or context of the game. In this spirit, it has also been attempted to generate music similar to a specific input to the degree that the original, to model upon, is presented to the generative part of the network, and not as input only during training ***(Koh et al. (C-RVAE), 2018)***.

3.2.4.5 Input Representation

In the early era, due to the often sparse number of training examples used, data used for learning seem to have been hand-crafted directly from examples. Thus, in this case, there was no separate input representation. From about the time when training using larger corpora became common, other types of input data surfaced among which, over the years, MIDI has been the most common one.

- **MIDI:**

  MIDI (Musical Instrument Digital Interface) was adopted in 1983 and is a standard way of communicating musical information in computers. A MIDI file contains a stream of events where two of the more significant events may be that a certain pitch is started (note on) or stopped (note off). However, there is a large variety of events and the events also contain information about what instrument is playing what, at what dynamic level (called velocity) and much more. MIDI in itself does not model music specifically according to notation standards and each event contains a number indicating how many pulses have passed since the last event. To control the speed of the music, MIDI holds information about how many pulses that corresponds to a quarter note as well as how many microseconds corresponds to a quarter note, thus a conversion between time, pulses and notated music is possible. Without restrictions during MIDI recording, the resulting music is not necessarily easily transformable into valid sheet music and given the granularity of the number of pulses per quarter note, micro-timing in the sense of being slightly late, or early, which is often the case in live music, is possible.

One of the absolutely earliest cases of using MIDI when modelling music with neural networks was in Neuro-Musician where the input of a jazz soloist, in real time, was fed to the model as MIDI and converted on the fly to the chosen data representation after which the output was converted back to MIDI and played on a speaker ***(Nishijima and Watanabe (Neuro-Musician), 1993)***. CHIME is a system that have followed in its footsteps since using the framework Sound2Midi for the live conversion ***(Franklin (CHIME), 2001)***. From the middle era, MIDI becomes more and more common ***(Eck and LaPalme, 2006; Adiloglu and Alpaslan (NeuroComposer), 2007; Coca et al., 2011)***. Franklin then uses both KeyKit ***(Franklin, 2006)*** and Band in the Box for MIDI input preprocessing ***(Franklin, 2006; Franklin, 2005)***. The MIDI trend then continues into the recent era ***(Boulanger-Lewandowski et al. (RNN-RBM, RNN-NADE), 2012; Goel et al. (RNN-DBN), 2014; Liu and Ramakrishnan, 2014; Sigtia et al., 2014; Fabius and van Amersfoort (VRAE), 2015; Bayer and Osendorfer (STORN), 2015; Gan et al. (TSBN), 2015; Lyu et al. (LSTM-RTRBM), 2015; Vohra et al. (LSTM-DBN), 2015; Choi et al. (char-RNN, word-RNN), 2016; Chu et al., 2016; Huang and Wu, 2016; Jaques et al. (RL Tuner), 2016; Lattner et al. (C-RBM), 2016; Madjiheurem et al. (Chord2Vec), 2016; Mogren (C-RNN-GAN), 2016; Sun et al., 2016; Walder, 2016; Bretan et al., 2017; Brunner et al. (JamBot), 2017; Colombo et al. (DAC), 2017; Johnson (LSTM-NADE, TP-LSTM-NADE, BALSTM), 2017; Dong et al. (MuseGAN), 2017; Guimaraes et al. (ORGAN), 2017; Hennig et al. (Classifying VAE, Classifying VAE+LSTM), 2017; Huang et al. (CocoNet), 2017; Jaques et al. (Sequence Tutor), 2017; Lee et al. (SeqGAN), 2017; Malik and Ek (StyleNet), 2017; Roberts et al. (MusicVAE), 2017; Roberts et al. (MusicVAE), 2018; Sabathé et al. (DRAW), 2017; Shin et al., 2017; Simon and Oore (PerformanceRNN), 2017; Teng et al., 2017; Tikhonov and Yamshchikov (VRASH), 2017; Ycart and Benetos, 2017; Yang et al. (MidiNet), 2017; Colombo and Gerstner (BachProp), 2018; Koh et al. (C-RVAE), 2018; Mao et al. (DeepJ), 2018; Simon et al. (MusicVAE), 2018; Walder and Kim (MotifNet), 2018)***, much thanks to the availability of data.

Because of the lack of restriction on durations in arbitrary MIDI, turning MIDI into a data representation is not always straightforward. One may record MIDI with a certain quantization, which means that during recording, the notes played will be adjusted to all start and end on the quantization unit, often chosen to be a sixteenth note or so. However, when no quantization is used, which is often the case for live performances. durations may start and end on such fine-grained units such as a 96th fraction of a quarter note or even smaller. Thus, some procedure must be in place for the conversion. A common scheme is to sample the MIDI file according to some time interval that corresponds to a minimum duration (likely to be the same as the chosen time slicing unit) ***(Boulanger-Lewandowski et al. (RNN-RBM, RNN-NADE), 2012; Fabius and van Amersfoort (VRAE), 2015; Yu et al. (SeqGAN), 2016)***. It is not intuitively straightforward that this will generate very accurate transcriptions however and other, more elaborate procedures have been devised as well.

- **ABC:**

  ABC is a textual representation of music which contains both meta data as well as the actual music. It has been used as input representation a few times ***(Boulanger-Lewandowski et al. (RNN-RBM, RNN-NADE), 2012; Goel et al. (RNN-DBN), 2014; Sigtia et al., 2014; Bayer and Osendorfer (STORN), 2015; Gan et al. (TSBN), 2015; Colombo et al., 2016; Colombo et al. (DAC), 2017; Sturm et al. (char-rnn, folk-rnn), 2016; Agarwala et al., 2017)***.

- **Humdrum:**

  Humdrum is a flexible textual format that allows a user to define the actual format to use themselves. Its structure holds information about what it encodes and it uses the vertical direction for successive events and the horizontal direction for simultaneous events, yielding a matrix-like format similar to how data would be modelled in an neural network. By specifying the format, time slicing or specific durations can be used. Even though not very common, it has been used in music modelling ***(O'Brien and Román (MusicNet), 2016)***.

- **MusicXML:**

  Being not as common as MIDI, the MusicXML format has been used as well, often when authors want to train on some specific music that doesn't necessarily exist in MIDI form. Examples are Scott Joplin ***(Sun (DeepHear), 2015)*** jazz soloists Charlie Parker, Miles Davis, Louis Armstrong ***(De Prisco et al., 2017)*** and of course J.S. Bach ***(Liang et al. (BachBot), 2017)***. Also general datasets in MusicXML have been distributed and subsequently used ***(Bretan et al., 2016; Wu et al. (HRNN), 2017)***.

- **Band in a Box:**

  A program rather than a format but its format has been used as well ***(Choi et al. (char-RNN, word-RNN), 2016)***.

- **Audio:**

  Using plain wave audio, or mp3, as input representation is not common but has been done in a few cases ***(Sarroff and Casey (DeepAutoController), 2014; Nayebi and Vitelli (GRUV), 2015; Kalingeri and Grandhe, 2016; van den Oord et al. (WaveNet), 2016a; Mehri et al. (SampleRNN), 2017; Engel et al., 2017)***. In one case, synthesized audio from MIDI files were used ***(Eppe et al., 2018)***.

- **Other:**

  Before the more common input representations got more or less standardized, different rare representations were used. One example is leadsheet representation in the format of a software called Impro-Visor ***(Bickerman et al. (RBM-provisor), 2010)***.

3.2.4.6 Data Representation

This section describes how data, on a low level, is represented when processed by the actual training algorithm. This is in contrast to input representation which describes the representation of music before any preprocessing steps occurring before the actual learning takes place. In some cases it is common to let a network determine an embedding for the initial low level data representation through the use of an embedding layer, in which case the output of this layer might be seen as the actual data representation ***(Huang et al. (ChordRipple), 2016; Huang and Wu, 2016; Madjiheurem et al. (Chord2Vec), 2016; O'Brien and Román (MusicNet), 2016; Walder, 2016; Agarwala et al., 2017; Brunner et al. (JamBot), 2017; Mehri et al. (SampleRNN), 2017; Lee et al. (SeqGAN), 2017; Liang et al. (BachBot), 2017; Tikhonov and Yamshchikov (VRASH), 2017)***. Such a representation is essentially distributed and absolute or relative, depending on what the input to the embedding layer is.

The most basic way to describe music symbolically on a low level is to model the properties of pitch and duration even though additional aspects to this exists. As far as pitch is concerned, no matter the representation, some range of represented pitches is chosen. Ranges can be a single octave ***(Gang and Lehmann, 1995; Gang and Berger, 1996; Large et al., 1995; Goldman et al. (NetNeg), 1999; Berger and Gang, 1997; Gang et al. (HNN), 1997; Berger and Gang, 1999; Gang et al. (HNN), 1999; Melo and Wiggins, 2003; Johnson (LSTM-NADE, TP-LSTM-NADE, BALSTM), 2017; Verbeurgt et al., 2004; Adiloglu and Alpaslan (NeuroComposer), 2007; Bickerman et al. (RBM-provisor), 2010; Tikhonov and Yamshchikov (VRASH), 2017)***, a few octaves ***(Todd, 1988; Todd, 1989; Freisleben, 1992; Tsang and Bellgard (EBM), 1992; Bellgard and Tsang (EBM), 1994; Franklin (CHIME), 2001; Eck and Schmidhuber, 2002; Melo and Wiggins, 2003; Eck and LaPalme, 2006; Fabius and van Amersfoort (VRAE), 2015; Bretan et al., 2016; Chu et al., 2016; Colombo et al., 2016; Jaques et al. (RL Tuner), 2016; Lattner et al. (C-RBM), 2016; O'Brien and Román (MusicNet), 2016; Sun et al., 2016; Bretan et al., 2017; Brunner et al. (JamBot), 2017; Huang et al. (CocoNet), 2017; Jaques***

et al. (Sequence Tutor), 2017; Wu et al. (HRNN), 2017; Yang et al. (MidiNet), 2017; Mao et al. (DeepJ), 2018), the (approximately) 88 keys of the piano ***(Boulanger-Lewandowski et al. (RNN-RBM, RNN-NADE), 2012; Goel et al. (RNN-DBN), 2014; Liu and Ramakrishnan, 2014; Sigtia et al., 2014; Gan et al. (TSBN), 2015; Lyu et al. (LSTM-RTRBM), 2015; Sun (DeepHear), 2015; Vohra et al. (LSTM-DBN), 2015; Madjiheurem et al. (Chord2Vec), 2016; Walder, 2016; Dong et al. (MuseGAN), 2017; Hennig et al. (Classifying VAE, Classifying VAE+LSTM), 2017; Ycart and Benetos, 2017; Colombo and Gerstner (BachProp), 2018)*** or the 128 pitches represented in the MIDI format ***(Liang et al. (BachBot), 2017; Malik and Ek (StyleNet), 2017; Roberts et al. (MusicVAE), 2017; Roberts et al. (MusicVAE), 2018; Sabathé et al. (DRAW), 2017; Simon and Oore (PerformanceRNN), 2017; Simon et al. (MusicVAE), 2018)***.

- **Pitch:**

  First of all, as part of all neural network learning problems, input data can be represented in binary form, discrete form and real form. The binary form is, as usual, often to prefer, and one-hot input is frequently used in music when modelling monophonic music. In polyphonic settings, a many-hot encoding is typically used. In a few cases, discrete and real numbers are used in different ways.

  Pitch might further be represented either in a local or distributed manner ***(Nierhaus, 2009)***, where, in the former, the activation of a pitch corresponds to a change in one input unit only. The alternative is to represent a pitch with a distributed representation using several input units each, for example $[1.0, 1.0, 0.0]$ may represent C whereas $[1.0, 1.0, 1.0]$ and $[0.0, 0.0, 1.0]$ represents D and E respectively. The disadvantage of a distributed representation is that it can give the network the illusion that some pitches are closer to each other than others. With our example encodings, C and D are, according to a lot of interpretations, closer than C and E, which might be desirable or not; it's not always the case that it is closer to the correct answer if the output of the algorithm picks a D instead of an E when it should be a C. A similar argument arises when expressing pitches as real numbers where, for example, C is 2.0, D is 3.0 and E is 4.0. Here C and E seem less alike than C and D and D and E which might be deceitful in some situations. On the other hand, one might argue against a representation where all input units are equally distanced from each other as well and claim that according to how we perceive pitch, both subjectively and physically in terms of overtone series, some pitches and chords are more similar to one another. Modelling pitches or chords according to this principle results in a distributed representation where extra thought has been put into the distances that arise between tokens. Such a representation has been referred to as "psychoacoustically motivated" ***(Laden and Keefe, 1989; Mozer (CONCERT), 1990; Mozer and Soukup (CONCERT), 1991; Mozer (CONCERT), 1994)*** and represents, for example, C with an encoding more similar to that of G than to that of F# with the motivation that G is a frequent pitch in the overtone series of C and thus appears together with (or in close vicinity to) C in tonal music to a much higher degree than F#. Thus, a network could be considered more correct in mispredicting a C for a G than for an F#. The inclusion of these principles is a way of guiding models in terms of data representation.

Pitches might also be represented either in an absolute or relative manner ***(Nierhaus, 2009)***. In the absolute case, a given input is always represented in the same way, no matter what came before it and no matter from what part of the network we are looking at it, whereas in a relative representation, the input depends on sequential or spatial context. An advantage of absolute representation is that the algorithm, during the creation phase, may make a single error but still get the subsequent notes correct; with a relative representation, an error may propagate and result in errors in the remaining part of the output as well ***(Todd, 1988; Todd, 1989; Adiloglu and Alpaslan (NeuroComposer), 2007)***. However, with a relative representation, we may span a large number of absolute pitches by using many small steps, thus with few input units ***(Corrêa et al., 2008)***, whereas with an absolute representation, no matter if it is local or distributed, the full state of the pitch part of the system, thus all input units, have to be represented in every input. This is inefficient since most inputs are sparse; the number of simultaneously active voices (pitches) are generally much lower than the number of possible pitches. Also, one thought here is that large leaps are uncommon in music generally, which may be taken advantage of by having fewer relative input units that represent the sizes of smaller leaps (intervals). Finally, modelling pitch absolutely implies that a network may be able to perfectly predict a given melody in one key, but not at all in another ***(Corrêa et al., 2008)***. The fact that two pieces, the second piece being a transposition of the first, sound about the same to most people is, as previously said, a property of music called transposition invariance ***(Johnson (LSTM-NADE; TP-LSTM-NADE, BALSTM), 2017)***, and with relative modelling, we get it for free. Two common ways to make up for pitch invariance with absolute representation are to either transpose all training data to a common key or to transpose all training data to all (or many) keys ***(Bickerman et al. (RBM-provisor), 2010; Bretan et al., 2016; Madjiheurem et al. (Chord2Vec), 2016; O'Brien and Román (MusicNet), 2016; Walder, 2016; Agarwala et al., 2017; Bretan et al., 2017; Hadjeres et al. (GLSR-VAE), 2017; Hadjeres and Pachet (DeepBach), 2017; Hadjeres and Nielsen (Anticipation-RNN), 2017; Simon and Oore (PerformanceRNN), 2017; Ycart and Benetos, 2017; Yang et al. (MidiNet), 2017)***. The former often implies that all training data is transposed to, or filtered for, a fixed pitch class (for example C major and C minor) ***(Boulanger-Lewandowski et al. (RNN-RBM, RNN-NADE), 2012; Lyu et al. (LSTM-RTRBM), 2015; Choi et al. (char-RNN, word-RNN), 2016; Chu et al., 2016; Huang et al. (ChordRipple), 2016; Huang and Wu, 2016; Sturm et al. (char-rnn, folk-rnn), 2016; Brunner et al. (JamBot), 2017; Johnson (LSTM-NADE, TP-LSTM-NADE, BALSTM), 2017; Hennig et al. (Classifying VAE, Classifying VAE+LSTM), 2017)*** or a fixed scale or key (for example C major and its relative A minor with the same pitch content) ***(Mozer (CONCERT), 1990; Mozer and***

Soukup (CONCERT), 1991; Mozer (CONCERT), 1994; Gang and Lehmann, 1995; Gang et al. (HNN), 1997; Gang et al. (HNN), 1999; Eck and LaPalme, 2006; Adiloglu and Alpaslan (NeuroComposer), 2007; Colombo et al., 2016; Sun et al., 2016; Liang et al. (BachBot), 2017; Shin et al., 2017) whereas the latter can be seen as a type of data augmentation and increases the size of the dataset by a factor of twelve and is frequently used as well. Yet another way of doing is to let input data remain in its original keys but transpose every sequence on the fly, randomly, during training ***(Roberts et al. (MusicVAE), 2017; Colombo and Gerstner (BachProp), 2018; Simon et al. (MusicVAE), 2018)***. However, just using a single key or scale might have implications for what music is allowed to model and in a piece that modulates, the episode in another key might suffer from this, especially if there are some other key specific considerations (such as the chords from C major in a local absolute encoding). A few simply uses the data as is and do not bother with the mentioned problem ***(Goel et al. (RNN-DBN), 2014; Liu and Ramakrishnan, 2014; Huang and Wu, 2016; Colombo et al. (DAC), 2017; Johnson (LSTM-NADE, TP-LSTM-NADE, BALSTM), 2017; Hennig et al. (Classifying VAE, Classifying VAE+LSTM), 2017)***. Worth to point out is that it is generally less straightforward how to model pitch successfully in a relative fashion.

The most common way to model pitch in networks working in the symbolic domain is to use a binary, local, absolute encoding. In such, there are n binary ordered input units used to represent the same number of ordered pitches. Having an input set to 1.0, indicates an active pitch and non-active pitches are set to 0.0. In some work (often early), the given range is limited to a few octaves ***(Todd, 1988; Todd, 1989; Freisleben, 1992; Tsang and Bellgard (EBM), 1992; Bellgard and Tsang (EBM), 1994; Franklin (CHIME), 2001; Eck and Schmidhuber, 2002; Melo and Wiggins, 2003; Eck and LaPalme, 2006; Fabius and van Amersfoort (VRAE), 2015; Bretan et al., 2016; Chu et al., 2016; Colombo et al., 2016; Jaques et al. (RL Tuner), 2016; Lattner et al. (C-RBM), 2016; O'Brien and Román (MusicNet), 2016; Sun et al., 2016; Bretan et al., 2017; Brunner et al. (JamBot), 2017; Huang et al. (CocoNet), 2017; Jaques et al. (Sequence Tutor), 2017; Wu et al. (HRNN), 2017; Yang et al. (MidiNet), 2017; Mao et al. (DeepJ), 2018)*** whereas recent work tend to model the full range of a piano (typically around 88 pitches) ***(Boulanger-Lewandowski et al. (RNN-RBM, RNN-NADE), 2012; Goel et al. (RNN-DBN), 2014; Liu and Ramakrishnan, 2014; Sigtia et al., 2014; Gan et al. (TSBN), 2015; Lyu et al. (LSTM-RTRBM), 2015; Sun (DeepHear), 2015; Vohra et al. (LSTM-DBN), 2015; Madjiheurem et al. (Chord2Vec), 2016; Walder, 2016; Dong et al. (MuseGAN), 2017; Hennig et al. (Classifying VAE, Classifying VAE+LSTM), 2017; Ycart and Benetos, 2017; Colombo and Gerstner (BachProp), 2018)***, leading to a so-called piano roll representation (matrix). This name is inspired from the piano rolls used in self-playing pianos in the early 20th century. In some cases, only one octave is used which corresponds to pitch class information only ***(Gang and Lehmann, 1995; Gang***

and Berger, 1996; Large et al., 1995; Goldman et al. (NetNeg), 1999; Berger and Gang, 1997; Gang et al. (HNN), 1997; Berger and Gang, 1999; Gang et al. (HNN), 1999; Melo and Wiggins, 2003; Johnson (LSTM-NADE, TP-LSTM-NADE, BALSTM), 2017) sometimes enhanced with a local input for octave making the representation somewhat distributed ***(Verbeurgt et al., 2004; Adiloglu and Alpaslan (NeuroComposer), 2007; Bickerman et al. (RBM-provisor), 2010; Tikhonov and Yamshchikov (VRASH), 2017)***. Yet another alternative is to model all the pitches of the MIDI format which are 128 in number ***(Liang et al. (BachBot), 2017; Malik and Ek (StyleNet), 2017; Roberts et al. (MusicVAE), 2017; Roberts et al. (MusicVAE), 2018; Sabathé et al. (DRAW), 2017; Simon and Oore (PerformanceRNN), 2017; Simon et al. (MusicVAE), 2018)***. A non-binary but local and absolute way of modelling pitch is to use numbers representing, for example, MIDI pitch numbers or piano key ordering, as is, in a discrete way ***(Madjiheurem et al. (Chord2Vec), 2016; Yu et al. (SeqGAN), 2016; Johnson (LSTM-NADE, TP-LSTM-NADE, BALSTM), 2017; Guimaraes et al. (ORGAN), 2017; Hadjeres et al. (GLSR-VAE), 2017; Hadjeres and Pachet (DeepBach), 2017; Hadjeres and Nielsen (Anticipation-RNN), 2017; Liang et al. (BachBot), 2017; Shin et al., 2017; Walder and Kim (MotifNet), 2018)*** or normalized to be real-valued in $[0, 1]$ ***(Mogren (C-RNN-GAN), 2016)***. In some cases, the actual note names are used, in which case the representations of, for example, C# and Db are different even though they are the same sounding pitch ***(Hadjeres et al. (GLSR-VAE), 2017; Hadjeres and Pachet (DeepBach), 2017; Hadjeres and Nielsen (Anticipation-RNN), 2017)*** (with MIDI numbers this distinction is not possible). This difference is due to context and differ depending on the key of the music which is helpful in models that deal with this distinction in the output. The discrete representation can also be used in a somewhat distributed way, as in the binary case, using pitch classes with additional information about octave. Such representation along with the equivalent representation of a pitch set for chords as well as duration has been used in a word-like manner ***(Lee et al. (SeqGAN), 2017)***.

Using symbolic music in textual form as input, a local absolute encoding often arises from describing the input pitches (or instruments in the case of percussion instruments) during a time slice as a class in a binary string with activities. In a melody, each such pitch may be represented as a word ***(Choi et al. (char-RNN, word-RNN), 2016; Sturm et al. (char-rnn, folk-rnn), 2016)*** or as a sequence of characters ***(Sturm et al. (char-rnn, folk-rnn), 2016; Agarwala et al., 2017)***, the latter effectively forming an absolute representation distributed in time. Such a representational string can be processed sequentially ***(Choi et al. (char-RNN, word-RNN), 2016)*** or as a single token at a time (in the resulting music) ***(Sturm et al. (char-rnn, folk-rnn), 2016)***. The latter would imply the opposite of a distributed encoding, that is, a single word token that describes the activities of several pitches, effectively forming a local, absolute chord representation instead. In the textual domain, pitches may also be encoded by their MIDI messages (separate on and off messages) ***(Huang and Wu, 2016)*** or by a string with a pitch number ***(Huang and Wu, 2016)***, both of which constitutes a local, absolute representation when used as tokens in a text processing RNN.

In a few works, a distributed, absolute pitch representation is used. One such "psychoacoustically motivated" representation still uses a binary vector to represent an ordered absolute range of pitches, but uses five activations to represent one actual pitch ***(Laden and Keefe, 1989)***. The idea is that a pitch is represented by the actual pitch along with activations in its five closest subharmonics, which is a series of pitches at different intervals below the sounding pitch. An approach with similar ideas, often referenced in the literature, uses thirteen activations to represent one pitch: six inputs represent the chromatic location (to account for chromatic vicinity), six inputs represent the location in the circle of fifths (to account for tonal vicinity) and the last input represent the pitch height originally in a logarithmic scale ***(Mozer (CONCERT), 1990; Mozer and Soukup (CONCERT), 1991; Mozer (CONCERT), 1994)*** but is also later used with MIDI numbers ***(Adiloglu and Alpaslan (NeuroComposer), 2007)***. In such a system, every pitch class (for example C) is represented with the same first twelve inputs, but differ in the pitch height input depending on which octave the pitch is in. Building on this concept, as well as an idea by Laden and Keefe ***(Laden and Keefe, 1989)***, a representation with seven bits, each corresponding to a circle of thirds, can be used to represent an octave of pitches ***(Franklin, 2004; Franklin and Locke, 2005; Franklin, 2006; Franklin, 2005; Coca et al., 2011)***. The idea here is that there are four circles of major thirds and three of minor thirds and one pitch occurs in exactly one major circle and one minor; setting these circle indices to 1 in the seven-bit vector is the representation of a certain pitch (class). Additional input for octave may augment this representation to cover multiple octaves ***(Coca et al., 2011)***. Chords may also be represented by their harmonic function (this assumes that all music is modelled with respect to one key since the absolute representation otherwise vary) in a local, absolute fashion whereby pitches are represented by a binary vector indicating in which chords the pitch occurs ***(Hild et al. (HARMONET), 1991)***.

Using a distributed, relative representation is unusual but one such example is MELONET which uses a representation that is relative to a reference note whereby subsequent notes are coded with a representation of interval and direction with an extra bit to add an interval of an octave ***(Hörnel and Menzel (MELONET), 1998)***.

Pitch may also be modelled in a local, relative fashion ***(Lewis (CBR), 1991; Nishijima and Watanabe (Neuro-Musician), 1993; Chen and Miikkulainen, 2001; Corrêa et al., 2008; De Prisco et al., 2017; Teng et al., 2017)***. In the case of monophonic or homophonic music, this can be done with integers indicating the number of steps (or interval) between the current pitch and the next ***(Lewis (CBR), 1991; Nishijima and Watanabe (Neuro-Musician), 1993; Chen and Miikkulainen, 2001; Corrêa et al., 2008; Teng et al., 2017)***. Note that this is the same as having an arbitrary reference note and choose the last seen note as this reference note. The reference can also be a chord against which the pitch corresponds to a step (not necessarily included in the chord but from the scale it is built on) ***(De Prisco et al., 2017)***. In the case of polyphonic music, an output may represent whether a specific pitch should be on or off for the next time step, given a vector of relative surrounding pitches as input ***(Johnson (LSTM-NADE, TP-LSTM-NADE, BALSTM), 2017; Mao et al. (DeepJ), 2018)***. This requires that each pitch is modelled separately to some extent, preferably by the same network.

A representation not resembling any other represents the contents of a unit of monophonic music, typically one or a few measures ***(Bretan et al., 2016)***, in a bag-of-words (BOW) manner. Here, the exact content of the unit is not represented, totally in line with BOW where words counts of sentences are accounted for but not the overall order. In this context, instead the number of occurrences of pitches and pitch bigrams of certain heights, the number and nature of intervals present and much more are accounted for. The representation used in this publication is indeed absolute but whether it is distributed or local is hard to tell, since several things are represented in parallel. One might argue, however, that such a representation is distributed.

- **Chords:**

  Not all models have the notion of chords as something clearly separated from melodic pitches, and we typically find this in architectures in the musical domains of harmonization and melody with chords. Many of these model chords in a distributed, absolute fashion as a group of pitches ***(Laden and Keefe, 1989; Tsang and Bellgard (EBM), 1992; Bellgard and Tsang (EBM), 1994; Gang and Berger, 1996; Berger and Gang, 1997; Berger and Gang, 1999; Eck and Schmidhuber, 2002; Melo and Wiggins, 2003; Franklin, 2004; Franklin and Locke, 2005; Franklin, 2006; Franklin, 2005; Bickerman et al. (RBM-provisor), 2010)*** or attributes such as root, inversion, extensions and alterations ***(Huang et al. (ChordRipple), 2016; De Prisco et al., 2017; Teng et al., 2017)*** or a combination thereof with pitch class and chord quality ***(Yang et al. (MidiNet), 2017)***. One representation used by Laden and Keefe was hinted at briefly in the previous section; the pitches of a chord are represented as the sum of the representation of their constituent pitches where each pitch is represented by the sounding pitch along with five of its closest subharmonic pitches ***(Laden and Keefe, 1989)***. Pitches that occur in several subharmonic series receive an input multiplied by the number of series they occur in. The third version of CONCERT, uses the very same representation but with small alterations, such as weighed subharmonic contribution (smaller weights the farther down in the subharmonic scale a pitch occurs) and not representing several octaves of pitches but only one, corresponding to all pitch classes ***(Mozer (CONCERT), 1994)***. One extra input based on similarity between chords is also included. Yet another representation inspired by both Mozer's pitch representation and the chord representation used by Laden and Keefe uses a representation with the sum of the circle of thirds representation of the individual pitches, to form the desired chord in question ***(Franklin, 2004; Franklin and Locke, 2005)***, later also with normalization so that all values lie in $[0.0, 1.0]$ ***(Franklin, 2006; Franklin, 2005)***. Chords can also be distributed in time, which is the case when modelling polyphonic music sequentially (note by note), whereby the notes of a chord are modelled as a sequence of single notes with some mechanism indicating when the current time step is finished ***(Mogren (C-RNN-GAN), 2016; Walder, 2016)***. Distributed in time is also the textual representation of chords as a sequence of characters where the model only processes one character at a time ***(Choi et al. (char-RNN, word-RNN), 2016)***.

When modelling chords in a local, absolute fashion, each chord is modelled as part of a vocabulary and as a result of this, new chords are not possible to generate. As vocabulary, all major and minor chords ***(Shin et al., 2017)***, sometimes completed with dominant chords ***(Gang and Lehmann, 1995; Gang et al. (HNN), 1997; Gang et al. (HNN), 1999)*** as well as augmented chords ***(Simon et al. (MusicVAE), 2018)*** have been used. Another approach is to use the harmonic function (tonic, subdominant, etc...) ***(Lewis (CBR), 1991)***, sometimes augmented to a distributed representation with local input that determines inversion as well as characteristic dissonances, which presumes that all input uses the same scale (otherwise harmonic functions differ between input sequences) ***(Hild et al. (HARMONET), 1991)***. Using all the chords found in the training data as vocabulary, or a subset thereof, is also common ***(Eck and LaPalme, 2006; Choi et al. (char-RNN, word-RNN), 2016; Chu et al., 2016; Brunner et al. (JamBot), 2017)***. In the textual input domain, concatenating pitch numbers into single words treated as tokens in a word vocabulary also constitutes a local and absolute encoding of chords ***(Huang and Wu, 2016)***.

One distributed, relative chord representation uses a one octave local pitch vector relative to a reference note, resulting in a representation that models several types of chords as well as inversions ***(Hörnel and Menzel (MELONET), 1998)***.

When summarizing the pitch content of a bar in a chordal representation, it is possible to use a pitch-class local representation where each pitch class is represented by the sum of fractions of the bar that the pitch class is active for ***(Melo and Wiggins, 2003)***.

- **Percussion:** Rhythmic instruments are typically modelled in some sense that fits how chords are represented, therefore, the reader is urged to see representation of percussion instruments as a representation of several pitches at once. An important difference is that rhythmic instruments can typically not play sustained notes, and therefore the whole notion of duration can be omitted.

- **Duration:**

  No matter the musical domain, the problem of modelling duration has almost exclusively been solved by using so-called time slicing, whereby the temporal aspect of music is solved by dividing the aspect of time in music into fixed-length slices, out of which the data for a sequential time step corresponds to one. Such a solution is good because it is straightforward and it allows for independent simultaneous events which might otherwise be hard to model. Typically, it is argued that the size of the time slice should correspond to the greatest common factor of all note durations in the music one is trying to model ***(Adiloglu and Alpaslan (NeuroComposer), 2007)***. Even though this makes sense, it is not always feasible and may impact learning in a negative way. It is thus common that finer time slices is sacrificed in favor of shorter sequences and an easier learning problem which, in turn, requires notes to be quantized to a minimum note length (according to the size of the time slice) which might be greater than they really are ***(Eck and LaPalme, 2006)***. One argument against time slicing in general is that long notes require several identical input time steps and it is reasonable to think that a network suffers from having to learn to deal with this ***(Mozer (CONCERT), 1994)***. It is also argued that time slicing encourages the model to repeat already turned on keys ***(Shin et al., 2017, Ycart and Benetos, 2017)*** and the finer the granularity of the time slicing, the more this behaviour is encouraged, leading to a lower network loss but not really implying better generation capabilities. An important decision then is to determine the granularity of the time slicing which implicitly also restricts what durations can be expressed in the network. Most often, either the quarter note ***(Hild et al. (HARMONET), 1991; Lewis (CBR), 1991; Gang and Berger, 1996; Berger and Gang, 1999; Hörnel and Menzel (MELONET), 1998; Corrêa et al., 2008; Boulanger-Lewandowski et al. (RNN-RBM, RNN-NADE), 2012; Choi et al. (char-RNN, word-RNN), 2016; Huang et al. (CocoNet), 2017)***, the eighth note ***(Todd, 1988; Todd, 1989; Eck and Schmidhuber, 2002; Melo and Wiggins, 2003; Eck and LaPalme, 2006; Boulanger-Lewandowski et al. (RNN-RBM, RNN-NADE), 2012; Huang and Wu, 2016; O'Brien and Román (MusicNet), 2016; Brunner et al. (JamBot), 2017; Johnson (LSTM-NADE, TP-LSTM-NADE, BALSTM), 2017; Hennig et al. (Classifying VAE, Classifying VAE+LSTM), 2017; Huang et al. (CocoNet), 2017; Koh et al. (C-RVAE), 2018)*** or the sixteenth note ***(Freisleben, 1992; Nishijima and Watanabe (NeuroMusician), 1993; Hörnel and Menzel (MELONET), 1998; Franklin (CHIME), 2001; Sun (DeepHear), 2015; Choi et al. (char-RNN, word-RNN), 2016; Chu et al., 2016; Jaques et al. (RL Tuner), 2016; Lattner et al. (C-RBM), 2016; Johnson (LSTM-NADE, TP-LSTM-NADE, BALSTM), 2017; Guimaraes et al. (ORGAN), 2017; Hadjeres et al. (GLSR-VAE), 2017; Hadjeres and Pachet (DeepBach), 2017; Hadjeres and Nielsen (Anticipation-RNN), 2017; Hennig et al. (Classifying VAE, Classifying VAE+LSTM), 2017; Huang et al. (CocoNet), 2017; Jaques et al. (Sequence Tutor), 2017; Liang et al. (BachBot), 2017; Malik and Ek (StyleNet), 2017; Roberts et al. (MusicVAE), 2017; Roberts et al. (MusicVAE), 2018; Teng et al., 2017; Wu et al. (HRNN), 2017; Ycart and Benetos, 2017; Yang et al. (MidiNet), 2017; Mao et al. (DeepJ), 2018)*** is chosen as the unit of a time slice even though the half-note has occurred as well ***(Tsang and***

Bellgard (EBM), 1992; Bellgard and Tsang (EBM), 1994; Gang et al. (HNN), 1997; Gang et al. (HNN), 1999), typically in harmonization contexts since harmonies tend to change less often. Extreme cases, such as using the 32nd note ***(Sabathé et al. (DRAW), 2017)***, the 32nd note triplet (resulting in twelve slices for each quarter note) exist ***(Bickerman et al. (RBM-provisor), 2010)*** as well as 24 slices per quarter ***(Bretan et al., 2017; Dong et al. (MuseGAN), 2017)***, which has the benefit that a lot of both triplet and duplet subdivisions are accounted for. When modelling meta information in the textual domain, it is possible to include the time slicing unit as a modelling parameter and thus make it variable ***(Sturm et al. (char-rnn, folk-rnn), 2016)*** or just make it an output parameter ***(Sturm et al. (char-rnn, folk-rnn), 2016; Agarwala et al., 2017)***. With time slicing comes also, as mentioned, the problem of distinguishing between successively repeated notes and a single longer note. Naively, these are modelled in the same way and therefore, some extra device is required. Often, this device is an extra number of inputs giving information about whether a note is rearticulated or held from before ***(Todd, 1988; Todd, 1989; Freisleben, 1992; Hörnel and Menzel (MELONET), 1998; Franklin (CHIME), 2001; Adiloglu and Alpaslan (NeuroComposer), 2007; Johnson (LSTM-NADE, TP-LSTM-NADE, BALSTM), 2017; Liang et al. (BachBot), 2017; Malik and Ek (StyleNet), 2017; Mao et al. (DeepJ), 2018)***. Some use this input only to prolong whatever pitch was active last, without supplying that actual pitch again ***(Bickerman et al. (RBM-provisor), 2010; Jaques et al. (RL Tuner), 2016; Guimaraes et al. (ORGAN), 2017; Hadjeres et al. (GLSR-VAE), 2017; Hadjeres and Pachet (DeepBach), 2017; Hadjeres and Nielsen (Anticipation-RNN), 2017; Roberts et al. (MusicVAE), 2017; Roberts et al. (MusicVAE), 2018; Teng et al., 2017; Wu et al. (HRNN), 2017)***. The opposite, an input that explicitly shuts off the previous note, can also be used but may be ambiguous for the network to interpret if it is only used sometimes and implicitly indicated at other times; typically playing a different note at consecutive time steps shuts off the first implicitly but is indicated explicitly, should the first note be followed by a pause or be repeated ***(Jaques et al. (RL Tuner), 2016)***. Another alternative is to halve the time slicing unit (for example, if one wishes to time slice on eighth notes, one does it on sixteenth notes instead) and let the second unit in every successive pair of sixteenth notes be turned on if a note is sustained and turned off if it is not ***(Eck and Schmidhuber, 2002)***. This, effectively, amounts to the same thing as adding an extra input for each possible pitch (or voice if there is a restriction on voices) modelling rearticulation with the benefit that we might model a finer note length at times, if needed. The downside however is that longer successively repeated notes are cut off just to signal that they are not sustained. Sometimes, the problem of rearticulation is simply ignored and either the mentioned distinction cannot be made whether necessary or not ***(Tsang and Bellgard (EBM), 1992; Bellgard and Tsang (EBM), 1994; Bretan et al., 2017; Ycart and Benetos, 2017)*** or some heuristic is applied to solve the problem ***(Eck and LaPalme, 2006; O'Brien and Román (MusicNet), 2016; Brunner***

et al. (JamBot), 2017; Sabathé et al. (DRAW), 2017; Yang et al. (MidiNet), 2017). Reducing rearticulated notes to shorter values (if possible) resolves the issue in part ***(Lattner et al. (C-RBM), 2016)***. Finally, a lot of recent publications do not address the issue clearly ***(Boulanger-Lewandowski et al. (RNN-RBM, RNN-NADE), 2012; Goel et al. (RNN-DBN), 2014; Liu and Ramakrishnan, 2014; Sigtia et al., 2014; Fabius and van Amersfoort (VRAE), 2015; Gan et al. (TSBN), 2015; Lyu et al. (LSTM-RTRBM), 2015; Sun (DeepHear), 2015; Vohra et al. (LSTM-DBN), 2015; Choi et al. (char-RNN, word-RNN), 2016; Johnson (LSTM-NADE, TP-LSTM-NADE, BALSTM), 2017; Dong et al. (MuseGAN), 2017; Hennig et al. (Classifying VAE, Classifying VAE+LSTM), 2017; Huang et al. (CocoNet), 2017; Koh et al. (C-RVAE), 2018)***. With most percussive instruments, this is not a problem since sounds cannot be sustained ***(Choi et al. (char-RNN, word-RNN), 2016)***.

The problem formulation when not using time slicing is easiest when modelling monophonic music where durations can easily be represented in a local ***(Chen and Miikkulainen, 2001; Colombo et al., 2016; Sun et al., 2016; Shin et al., 2017; Tikhonov and Yamshchikov (VRASH), 2017)*** or distributed fashion ***(Mozer (CONCERT), 1990; Mozer and Soukup (CONCERT), 1991; Mozer (CONCERT), 1994; Huang and Wu, 2016; Colombo et al. (DAC), 2017)*** by the use of extra inputs. This, in turn, can either be done in conjunction with pitch modelling ***(Franklin, 2005; Huang and Wu, 2016; Sun et al., 2016; Tikhonov and Yamshchikov (VRASH), 2017)*** or in an entirely separate network ***(Mozer (CONCERT), 1990; Mozer and Soukup (CONCERT), 1991; Mozer (CONCERT), 1994; Franklin, 2006; Colombo et al., 2016)***. When modelling melody and chords, a word-like representation with variable explicit duration has been used in which duration, chord and melody form a single word ***(Lee et al. (SeqGAN), 2017)***. In the polyphonic case, should one model multiple pitches at the same time, using the same device becomes cumbersome since each pitch would have to have its own duration input which effectively doubles the number of inputs. It also leads to situations where only one voice, for example, changes while the others are sustained which would be represented by a new event where the sustained notes seem turned off, nonetheless, they would still sound from the previous event. It is hard to prove if this is a good or a bad thing but the idea that it seems unintuitive and "messy" is not far-fetched. If one models polyphony note by note (sequential polyphony, as opposed to parallel polyphony), for example by the inclusion of a relative timing mechanism that, for each event, indicates how far after the previous event it takes place, it is possible to model simultaneous notes sequentially by using the same local absolute duration that may be used for monophony ***(Mogren (C-RNN-GAN), 2016; Walder, 2016; Colombo and Gerstner (BachProp), 2018)***. This may seem as unintuitive as the previous suggestion however not as "messy" since events don't partly overlap in the same way (in the previous way, it might explicitly look as if the network is told that a note is turned off whereas in the latter case, the note in question will not be specifically mentioned since the whole range of notes is not explicitly modelled in every event).

In CONCERT, rhythm is modelled in a distributed way similar to how pitch is represented ***(Mozer (CONCERT), 1990; Mozer and Soukup (CONCERT), 1991; Mozer (CONCERT), 1994)***. A duration is modelled in terms of 1/12 of a quarter note. This measure is divided on two circles (in analogy with the pitch representation): the circle of 1/3 with values 0/12, 1/12, 2/12 and 3/12 and the circle of 1/4 with values 0/12, 1/12, and 2/12 ***(Mozer (CONCERT), 1994)***. Each circle is represented with two values and a fifth value is used to represent the logarithm of the duration measuring between 0/12 and 96/12 to express a full note (here corresponding to four quarter notes). This system seems much more complicated than most others but the benefits of it is that almost all note values may be expressed, even triplets and sextuplets. Inspired by Mozer and his abiliy to express a great many durations, Frankling derives a 16-bit representation that builds on the notion of a quarter note being divided into 96 slices (the default for MIDI) ***(Franklin, 2006; Franklin, 2005)***. The different indices in the representation corresponds to a different number of slices (the first index corresponds to 384 slices, equivalent to four quarter notes, or a whole note and the last to 1 slice, corresponding to $\frac{1}{96}$ of a quarter note) and each bit is set to 1 wherever the remainder, after doing the same procedure for previous indices, is >= 1 when divided by the number of slices corresponding to the index, in which case only the remainder is propagated further down. This creates a very fine-grained, yet dense, representation for a large number of durations. Franklin argues that one problem with the representation is that durations that are very similar often have very different representations.

A final suggested alternative uses a sort of time slicing but together with an absolute representation specifying the number of time slicing units to skip forward by the index of the turned on unit in a one-hot duration vector of a fixed size. Each unit in increasing order means that 10 ms. of actual time is added to the accumulated time of the last index, starting with 10 ms. and ending at 1 s. ***(Corrêa et al., 2008; Simon and Oore (PerformanceRNN), 2017; Simon et al. (MusicVAE), 2018)***. In polyphony, this may be used together with a direct use of the MIDI on and off messages ***(Simon and Oore (PerformanceRNN), 2017; Simon et al. (MusicVAE), 2018)***. A closely related way of doing it uses a counter that, using a one-hot encoding, increases when a note is sustained and resets when it is to be rearticulated ***(Chu et al., 2016)***.

As can be read in the pitch subsection, music has been represented as the content of a musical unit in a bag-of-words (BOW) manner, typically one or a few measures, and as such, duration has been a part of it ***(Bretan et al., 2016)***. This includes, among other things, counts of different fixed durations and duration bigrams.

Duration has also been distributed in time, by the MIDI on and off messages (tokens) for a pitch in the textual input domain ***(Huang and Wu, 2016)***.

Finally, in PerformanceRNN, the input dataset is not only augmented by transposition but also by increasing and decreasing all the note lengths of a piece of music by a fixed value ***(Simon and Oore (PerformanceRNN), 2017)***.

- **Signal:**

  When music is modelled from an audio signal, the signal is not always unaltered and a common approach is to transform it into spectrograms with Fast Fourier Transform (FFT) ***(Sarroff and Casey (DeepAutoController), 2014)***, Discrete Fourier Transform (DFT) ***(Nayebi and Vitelli (GRUV), 2015)*** or some other Fourier Transform (FT) ***(Kalingeri and Grandhe, 2016)*** and model these. Transformation into Mel spectrograms has been done as well ***(Eppe et al., 2018)***. If modelled as unaltered audio, one might reduce the size of each sample from 16 bits to 8 bits using PCM encoding ***(van den Oord et al. (WaveNet), 2016a; Mehri et al. (SampleRNN), 2017; Engel et al., 2017)***.

- **Conditioning / additional inputs:**

  Much additional data can be supplied to a model for various reasons. Here, we focus on additional inputs that are not notes, chords or some representation thereof, that some models use during generation to condition the output as part of the integral architectural decisions. For example, one might associate every input pattern with an identification (or plan) input that is held fixed for a certain pattern during training. By interpolating between, or extrapolating from, these plans the model can generate new material in the composition phase ***(Todd, 1988; Todd, 1989; Freisleben, 1992; Gang and Lehmann, 1995; Gang et al. (HNN), 1997; Goldman et al. (NetNeg), 1999; Gang et al. (HNN), 1999; Franklin (CHIME), 2001; Franklin, 2005)***. Sometimes supplying the composer or genre as additional inputs throughout a piece leads to interesting results ***(Bretan et al., 2017; Mao et al. (DeepJ), 2018)***. Another popular approach is to supply a model with the current beat ***(Gang and Berger, 1996; Berger and Gang, 1997; Gang et al. (HNN), 1997; Berger and Gang, 1999; Gang et al. (HNN), 1999; Bickerman et al. (RBM-provisor), 2010; Brunner et al. (JamBot), 2017; Johnson (LSTM-NADE, TP-LSTM-NADE, BALSTM), 2017)*** or sub-beat ***(Hadjeres and Pachet (DeepBach), 2017; Shin et al., 2017)***, whether the current beat is stressed ***(Hild et al. (HARMONET), 1991)*** or where in the musical phrase ***(Hild et al. (HARMONET), 1991; Tsang and Bellgard (EBM), 1992; Bellgard and Tsang (EBM), 1994; Gang et al. (HNN), 1999; Hörnel and Menzel (MELONET), 1998; Franklin, 2006; Franklin, 2005; Chu et al., 2016; Sturm et al. (char-rnn, folk-rnn), 2016; Hadjeres and Pachet (DeepBach), 2017; Liang et al. (BachBot), 2017)***, the current time step is, for the model to better learn structure and connect it to the metric properties of the time signature. The latter requires extra annotations or an additional subnet since the continuation of musical phrases is not possible to decide from data only. Supplying the time signature, often as a fixed input throughout a training pattern, has also been done ***(Mozer (CONCERT), 1990; Mozer and Soukup***

(CONCERT), 1991; Mozer (CONCERT), 1994; Sturm et al. (char-rnn, folk-rnn), 2016) as well as supplying the key (or tonic note) in the same manner ***(Adiloglu and Alpaslan (NeuroComposer), 2007; Chu et al., 2016; Sturm et al. (char-rnn, folk-rnn), 2016; Hadjeres and Pachet (DeepBach), 2017; Hennig et al. (Classifying VAE, Classifying VAE+LSTM), 2017)***. Others have also specified the time step in a piece ***(Walder, 2016)*** as well as specifically added inputs indicating whether a note is modelled on the same time step as the one before (in the case of modelling polyphony sequentially) ***(Walder, 2016)*** as well as whether notes have been turned off from the previous time step ***(Walder, 2016)***. Modelling rests, the lack of any pitches, as a special input is also common ***(Mozer (CONCERT), 1990; Mozer and Soukup (CONCERT), 1991; Mozer (CONCERT), 1994; Tsang and Bellgard (EBM), 1992; Bellgard and Tsang (EBM), 1994; Franklin (CHIME), 2001; Bickerman et al. (RBM-provisor), 2010; Choi et al. (char-RNN, word-RNN), 2016; Colombo et al., 2016; Jaques et al. (RL Tuner), 2016; Guimaraes et al. (ORGAN), 2017; Roberts et al. (MusicVAE), 2017; Roberts et al. (MusicVAE), 2018; Teng et al., 2017; Wu et al. (HRNN), 2017)***. Conditioning (and modelling) musical contour has been done in terms of inputs corresponding to fixed sizes of intervallic leaps to next pitch ***(Goldman et al. (NetNeg), 1999; Hörnel and Menzel (MELONET), 1998)*** as well as inputs modelling whether a melody is ascending or descending ***(Large et al., 1995; Goldman et al. (NetNeg), 1999; Hörnel and Menzel (MELONET), 1998; Chu et al., 2016)***. This is similar to modelling tension as a real value in $[0.0, 1.0]$ as an extra input, also requiring extra annotations ***(Melo and Wiggins, 2003)***. Abstract motive classes has been used to condition the elaboration of quarter notes into four sixteenth notes ***(Hörnel and Menzel (MELONET), 1998)***.

After training, the absolute largest part of all publications initialize the composition, or generation, phase by seeding the network with a few notes, chords or whatever it might be. A few works instead uses a special start (and subsequently an end) symbol instead ***(Todd, 1988; Todd, 1989; Adiloglu and Alpaslan (NeuroComposer), 2007; Bretan et al., 2016; Choi et al. (char-RNN, word-RNN), 2016; Colombo et al., 2016; Madjiheurem et al. (Chord2Vec), 2016; Sturm et al. (char-rnn, folk-rnn), 2016; Walder, 2016; Agarwala et al., 2017; Hadjeres and Nielsen (Anticipation-RNN), 2017; Liang et al. (BachBot), 2017; Shin et al., 2017)***. It has been tested whether supplying meta data about the song as initial state in an RNN results in good performance ***(Agarwala et al., 2017; Tikhonov and Yamshchikov (VRASH), 2017)*** but it was suggested that the information was ignored ***(Agarwala et al., 2017)***.

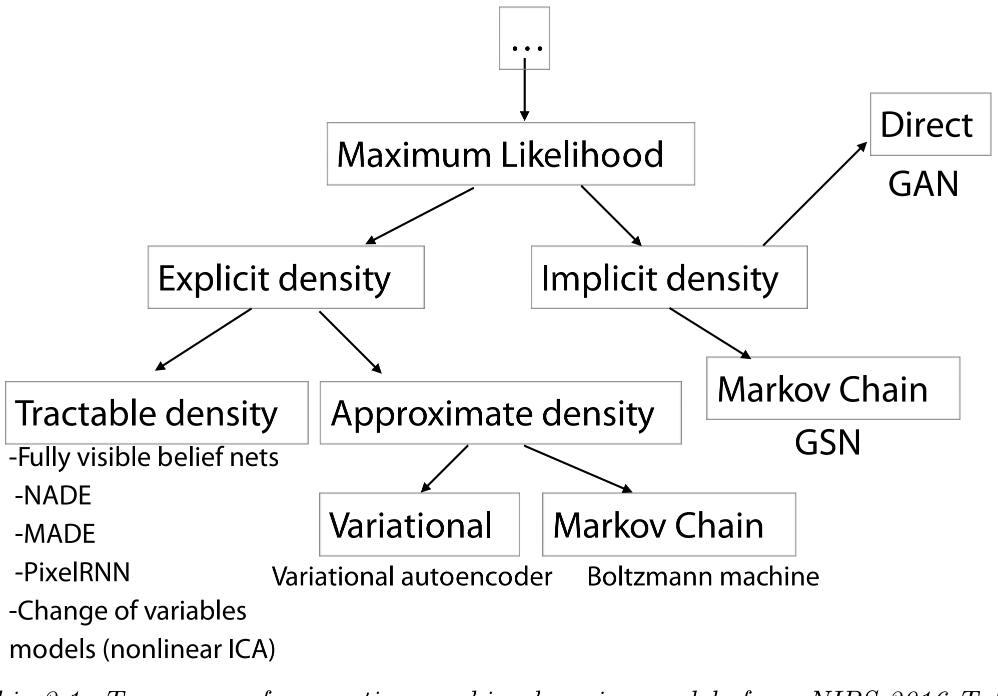
An occurring pattern of conditioning is also to supply so-called time-delay connections from previous time steps, effectively making the outputs from these time steps available for later predictions ***(Eck and LaPalme, 2006; Chu et al., 2016; Wu et al. (HRNN), 2017)***, thus facilitating the model's potential to discover repeated patterns, should one know at what time step intervals it is likely to find repeated patterns.

Some publications focuses entirely on aspects of notes that seem peripheral in the more general case, such as dynamics, or velocity as it is also called ***(Malik and Ek (StyleNet), 2017)***, and micro-timing (the offset between when a note should really have started or ended according to a strict interpretation of a written score and when a performer chooses to play it). Yet others model these things along with the more general division of pitch and duration ***(Nishijima and Watanabe (Neuro-Musician), 1993; Mogren (C-RNN-GAN), 2016; Simon and Oore (PerformanceRNN), 2017; Mao et al. (DeepJ), 2018; Simon et al. (MusicVAE), 2018)***.

More imaginative examples uses conditioning on extra-musical inspiration. For example, nature landscapes as a curve in 2D ***(Corrêa et al., 2008)*** or chaotic melodies taken from, among other things, solutions to differential equations ***(Coca et al., 2011)***.

3.2.4.7 Model / Architecture

This section is devoted to the kind of neural network architecture that has been used and also attempts to give a short overview of each publication in terms of info that does not fit anywhere else. Note that a model can contain several architectures in one and can thus be listed in several model categories. However, each model is necessarily listed under the most noticeable architecture used but only under other categories as well if there is anything particular to mention about it. To make it easier to overview different classes of models and how they relate to each other, it is a good idea to keep the taxonomy of generative models, as laid forth by the "GANfather" Ian Goodfellow ***(Goodfellow, 2017)***, in mind.



Graphic 3.1: Taxonomy of generative machine learning models from NIPS 2016 Tutorial: Generative Adversarial Networks by Ian Goodfellow. The typical use of vanilla neural networks as sequence predictors places them in the "tractable density" category.

- **Feedforward Neural Network (FFNN) or Multilayer Perceptron (MLP):**

  Feedforward models have been used sparsely but throughout the entire history of music modelling with neural networks. An early example is a 3-layer network used to classify chords ***(Laden and Keefe, 1989)***. The authors represent pitch in different ways and conclude that a distributed representation performs best. However, the compared setups use different amounts of training data making the comparison slightly unfair.

  Another early example of feedforward networks uses a 3-layer network to determine the next chord given the three preceding chords as well as the melodic context of the chord to predict ***(Hild et al. (HARMONET), 1991)***. Time slicing on quarter notes is used and after predicting the chord, another feedforward network ornaments the chordal structure with eighth notes.

  A model called CBR (Creation By Refinement) uses a 4-layer critique network to classify a musical sequence according to some criteria (typically binary indicating if the music is plainly good or bad) ***(Lewis (CBR), 1991)***. After training, the learned weights are held fixed along with an output label and the model successively alters randomized input until it has converged to an input pattern that adheres to the given output label.

  Also using feedforward networks, Neuro-Musician learns to respond to a multiple measure jazz solo in a way that is consistent in style ***(Nishijima and Watanabe (Neuro-Musician), 1993)***. The model uses several networks and starts by modelling contour, which is a reduced description of the melodic content over a few measures. The resulting contour is then matched with an indication of the note density in the passage after which several networks model a few measures each. The output consists of measures of rhythmic, dynamic and pitch content which is the final output after the rhythmic content is refined by a network that models expressive timing. All in all, the network learns how to model a solo based on the melodic skeleton along with the density of the input and when used after training, the idea is that it should manage to alternate in real time with a jazz player, answering the solos of the latter.

  Inspired by HARMONET, Hörnel and Menzel successfully trains several HARMONETs to recognize different chorale styles and use that as the starting point for their new architecture ***(Hörnel and Menzel (MELONET), 1998)***. MELONET uses a HARMONET to harmonize a quarter note chorale after which three separate feedforward networks model elaboration of the quarter notes to groups of four sixteenth notes in Pachelbel chorale style. The authors refer to these groups as motives and mean that there are melodic motives, that have small leaps and dissolve dissonances, and harmonic motives, that have larger leaps that outline a harmony. The networks work on different hierarchical levels where the first network, called a supernet, learns to predict a motive class based on the quarter note skeletal context as well as current harmony and most recent motive class along with a few other things. The chosen motive class is then forwarded to another network, called the subnet, that models the actual motive given the current and next quarter notes and current harmony. The new motive is then classified by the motive classifier and forwarded to the supernet for the next quarter note to embellish. RProp ***(Riedmiller and Braun, 1993)*** was used during training as opposed to backpropagation. The model is successful with test panels and is elaborate in its construction.

Using tension as an extra input to model chords off of the fourth movement of Prokofiev's classical symphony has been done with feedforward networks ***(Melo and Wiggins, 2003)***. The authors define tension in music as "... how unfinished the piece of music would sound if it stopped at that point". The tension profile has a value for every bar and in the first phase, one network models the next bar chord along with its tension given a number of chords preceding it. A chord is recorded from each bar in pitch class representation by summing the fraction of the bar a given pitch class (sum is over individual pitches in the pitch class) is sounding. In the next phase, another feedforward network produces a stream of eighth notes based on a previous window of eighth notes along with the current tension profile. The last network outputs a piano roll representation and can, in principle, model polyphonic music without restrictions. A threshold for when a note is to be included or not is derived based on how it affects the accuracy when compared to the original training data. The tension profile is acquired by averaging over annotated tension profiles from a number of test individuals. The authors are somewhat disappointed and state that a test panel thought that the music "sounded like random music, while admitting that some 'order' and even 'style' could be noticed".

Using a suffix tree to extract patterns, in terms of intervals, and using a Markov model to transition between these derived motivic patterns, a feedforward network has also been used as a subcomponent only to decide the starting pitch and duration of the pattern sampled from the Markov model ***(Verbeurgt et al., 2004)***.

Using feedforward networks as the main architecture became less and less common as the recurrent networks (and more complicated architectures) grew more popular and in one of the last publications to use a feedforward network only, NeuroComposer models first species counterpoint, also known as note-against-note counterpoint ***(Adiloglu and Alpaslan (NeuroComposer), 2007)***. The authors give a great resume over different methods to represent music (and even gives a fresh, coined, terminology to these) and arrive at using different representations for music when modelled as input and output (to the actual network. not to be mistaken for the input representation that is otherwise referred to as input which is the representation before preprocessing). NeuroComposer composes a counter-melody to a given melody using a sliding windows technique, as the authors call it, and takes the three last and the current note of the melody along with the three last notes of the counter-melody to compose the next note of the counter-melody. In the input, a pitch representation almost identical to Mozer's ***(Mozer (CONCERT), 1990; Mozer and Soukup (CONCERT), 1991; Mozer (CONCERT), 1994)*** is used for the seven different pitches whereas in the output, pitch classes along with an octave specification output node is used to predict the next note of the counter-melody. Both counter-melodies above and below the given melodies are learned and since the two voices have the same, monotonous, rhythmic pattern no further device for durations is needed than a rearticulation node for every note, indicating whether it was held from before or not. Even though the network learns some structural properties of the input data, the output contains mistakes and sounds uninteresting at times, the latter a property not rare in first species (note-against-note) counterpoint in general.

Even though using feedforward networks only grew less and less common, feedforward networks have occasionally been used as a subcomponent in larger systems based on the use of multiple different neural network architectures for different subtasks ***(Bretan et al., 2016; Mehri et al. (SampleRNN), 2017; Hadjeres and Pachet (DeepBach), 2017; Hennig et al. (Classifying VAE, Classifying VAE+LSTM), 2017; Walder and Kim (MotifNet), 2018)***.

One late use of feedforward networks is ChordRipple which uses word2vec along with the skip-gram model from linguistics adapted to chords ***(Huang et al. (ChordRipple), 2016)***. The skip-gram model uses learnable embeddings (word2vec implies that each word is embedded with learnable weights in a vector) for each word in the input vocabulary and adapts these during learning based on what the surrounding words, according to some window size, are. The corresponding method in music implies embedding chords and adjusting this embedding according to what the surrounding chords might be. By using cosine similarity, after learning, between learned embeddings one might find substitute chords that usually appear in the same context. One might also use the network to get suggestions for surrounding chords given a chord. With this principle, the authors behinds ChordRipple build an entire tool for reinventing and varying chord progressions which they try out on an audience of composers. With the tool, one might replace a single chord, or replace a whole context of chords, something that the authors call a rippling effect of changing a single chord. The tool is appreciated and the authors show that the embedding layer arranges the chords into the circle of fifths when visualized with dimensionality reduction techniques.

Also interested in modelling chord embeddings using a skip-gram model, Chord2Vec uses several models with embedding layers to see which performs best ***(Madjiheurem et al. (Chord2Vec), 2016)***. The simplest model is a feed-forward network with an embedding layer that should output the next and previous chords given an input chord. As expected this model performs worse than NADE-inspired and LSTM-based models. More can be read in the LSTM subsection.

- **Simple Recurrent Neural Network (SRN):**

  A Jordan network was used in one of the very first attempts by Todd to model monophonic music in which learning took place after each time step ***(Todd, 1988; Todd, 1989)***. The Jordan network generalized to new compositions after training by setting the plan units associated with each input patterns to new, unseen, patterns which led the network to, after being seeded with one or a few notes, compose melodies of its own. Inspired by Todd's approach, Freisleben extends it by adding what is called a sequential memory where only the first context unit receives the last step outputs. In the time step to come, it forwards its most recent state to the second context unit after which it receives new data from the model output ***(Freisleben, 1992)***. The sequential context is used along with the same, parallel (called exponential), context that is used by Todd as well with the motivation that the network should handle musical repetitions better. Several sets of output units are used working with the same two sets of context units to form polyphonic music.

For a series of publications, sometimes together with others sometimes alone, Gang uses and refines Jordan networks along three lines of experiments.

In the first series of attempts, Gang and Lehmann (and later Wagner) studies harmonization of melodies in a model later called a Human Neural Network (HNN) ***(Gang and Lehmann, 1995; Gang et al. (HNN), 1997; Gang et al. (HNN), 1999)***. First a feedforward subnet is used that take as input the collective pitch content of a window of music (typically a measure) and then outputs key notes of the window ***(Gang and Lehmann, 1995)***. These are then used as additional inputs to the Jordan network with two hidden layers where the recurrent input units corresponds to chords. The main hidden units model an octave of twelve pitches that are then partially connected to the 14 output units corresponding to the seven triads of C major along with the seven extended seventh chords of the same. Later, the model is extended to work in real time whereby harmonization is done simply by feeding the network notes, on the first and third beat only, sequentially while predicting harmonies ***(Gang et al. (HNN), 1997)***. At this point, the main sequential net has a subnet modelling meter as alternating between the first and third beat of the network incorporated. This setup is then further enriched with inputs corresponding to location in a phrase as well as context memory in both melody and harmonies, as opposed to in the harmonic progression only, as was the case earlier ***(Gang et al. (HNN), 1999)***.

In the second series of setups, sequences of harmonies are investigated with the purpose to model human musical cognition and the Degree of Realized Expectation (DRE) ***(Gang and Berger, 1996; Berger and Gang, 1997; Berger and Gang, 1999)***. Berger and Gang model harmonies as combinations of pitches in a network conditioned on beat. The model is taught harmonic progressions and is then fed progressions that are surprising in some way (either due to metric position or pitch content) after which the authors model the cognitive experience of musical surprise in terms of the network prediction probabilities and how these vary as a function of the DRE. Both duple meter ***(Gang and Berger, 1996)*** and duple and triple meter in parallel ***(Berger and Gang, 1997; Berger and Gang, 1999)*** is tried in order to account for how meter and beat affects harmonic expectation. In the latter, expectations on meter is also formalized as an output of the network. The authors argue that meter and harmonic sequences create latent expectations, even with theoretically inexperienced listeners, but suggest that determining the next chord as a negotiation between harmonic and metric state only may be somewhat ambiguous. An underlying conclusion of the contribution of these works is a philosophical one dealing with compositional strategies concerning how to know when to surprise and when not, and the authors conclude by citing Strunk in From Mozart to Stravinsky: "... to play on expectation he must first arouse it. To secure emphasis he must first exercise self control. He cannot afford to be continually surprising to his listener. He must be simple before he is complex, regular before he is irregular, straightforward before he is startling".

Finally, a hybrid approach with a utility-based agent composing counterpoint is suggested called NetNeg ***(Goldman et al. (NetNeg), 1999)***. Here, Gang teams up with Goldmann et al. modelling first species counterpoint, also known as note-against-note counterpoint. Two agents use a utility function to negotiate their next notes from activations given by a Jordan network modelling both melodies simultaneously, after which the decision of the agents is input for the next step predictions. The network models melodic contour in terms of intervallic leaps and melody direction to improve the results.

In the CHIME (Computer Human Interacting Musical Entity) model, a Jordan network models jazz melodies conditioned on chords with the purpose to trade jazz solos in real time ***(Franklin (CHIME), 2001)***. The network is first trained traditionally on some jazz solos, then weights are added and the trained model is trained anew with actor-critic reinforcement learning. At the point of performance, the input to the network is the solo played by the live player that the network is supposed to construct a variation of.

Elman networks were also used early on, for example in the CONCERT architecture where a network is trained to produce Bach melodies ***(Mozer (CONCERT), 1990; Mozer and Soukup (CONCERT), 1991; Mozer (CONCERT), 1994)*** using a "psychoacoustically motivated" pitch representation. Long-term structure is taken care of by experimenting with setting the tendency to forget previous context to different values in the context units, thus partitioning them into segments which, at the extreme, remembers over a long time or a short time. An Elman network has also been used and trained with a genetic algorithm according to a sum of weighed fitness functions that supposedly model music according to the style of Bartók as well as to general music theory rules ***(Chen and Miikkulainen, 2001)***.

One of the first uses of a model referred to as a recurrent neural network only, with no reference to Jordan or Elman networks, leaves a lot of details unexplained ***(Corrêa et al., 2008)***. A novelty in the work is that nature is used as inspiration during composition of melodies, which is done by taking a contour from a landscape and turning it into a curve in 2D with values fed to the network which predicts melodies, measure by meaure, with a relative representation of intervals. A shortcoming of the publication is that it is not clear whether a Jordan network or an Elman network is used and at times, it even seems as if a feedforward network is used where the output of one iteration is used as input to the next (thus without any recurrent units holding context). However, the input units are specifically mentioned as recurrent which indicates that it might be a Jordan network, but the recurrent input seems to come from the last step output which suggests that the network might not have any recurrent connections at all. It is also unclear whether the landscape contour is fed as a whole during composition, in which case it resembles a plan, or as a sequence. Finally, it is not known whether landscapes are used during training or not and, if so, if the curves are generated from music or taken randomly (to associate music with arbitrary curves, much like a plan input again). The idiomatic use of the network (with extra input) as well as the time leads to a classification as a Jordan network but as said, this remains unclear. Inspiration from nature reveals an imaginative approach, however, that along with the use of Gaussian noise in the input units are the biggest contribution of the work. The latter is used to combat the bad aspects of teacher forcing and makes the network exposed to more various input during training. The risk is otherwise that the use of teacher forcing leaves the network unexposed to certain inputs that might occur during composition (when teacher forcing is no longer applied) and which might lead to unexpected and degenerate behaviour. A self-organizing map (SOM) network is also tried out for composition and judging from the results, the latter performs poorly with large interval jumps constantly in a very unmusical way. The first approach however leads to melodies that seem more music-like. The publication also mentions that their model is trained with backpropagation through time (BPTT) along with online learning in a conflicting way.

One of the last references to an Elman network in music modelling uses a source of chaos as an additional input, similar to how plan units ***(Todd, 1988; Todd, 1989; Freisleben, 1992; Gang and Lehmann, 1995; Gang et al. (HNN), 1997; Goldman et al. (NetNeg), 1999; Gang et al. (HNN), 1999; Franklin (CHIME), 2001; Franklin, 2005)*** or nature landscapes ***(Corrêa et al., 2008)*** have been used. The chaos is an additional melody with a varying number of notes that is derived from different aspects of the training melody as, for example, solutions to first order differential equations arising from different measures. The chaotic melody is used as additional input during training, which is done with one melody only, after which the trained network should generate a melody of its own. Attempts are made with a varying number of notes in the chaotic melody and the authors, arguing that originality and complexity in music is related to a certain degree, show that their network manages to produce melodies that are similar compared to the original but still complex and therefore original and qualitative. Several measures are devised to express similarity and difference in melodic complexity between melodies and comparisons are made when training the network with none up to twenty notes in the chaotic melody. The publication is vague on some points and it is unclear what representation the authors use for duration as well as several details regarding how the chaotic melody is presented to the network. One might also consider it self-evident that the trained model will produce output more dissimilar to the input melody the more additional content, e. g. the longer the chaotic melody, it has been trained with.

- **Recurrent Neural Network (RNN):**

  Many later uses of RNNs (as well as LSTMs and GRUs) only use the model to account for temporal context only, as opposed to also using its hidden state and one or more fully-connected layers to generate output ***(Boulanger-Lewandowski et al. (RNN-RBM, RNN-NADE), 2012; Goel et al. (RNN-DBN), 2014)***. These models generally have some other architecture to generate the actual outputs, such as an RBM ***(Boulanger-Lewandowski et al. (RNN-RBM, RNN-NADE), 2012)***, a NADE ***(Boulanger-Lewandowski et al. (RNN-RBM, RNN-NADE), 2012; Sigtia et al., 2014)***, a DBN ***(Goel et al. (RNN-DBN), 2014)*** or a VAE ***(Bayer and Osendorfer (STORN), 2015)***. Another approach is to use an RNN as a baseline, or reference, when trying out more sophisticated models ***(O'Brien and Román (MusicNet), 2016)***.

  When using a music model to impose contraints and guidance in Automatic Music Transcription, an RNN is trained to act as a prior, trained with both stochastic gradient descent (SGD) and Hessian free optimization (HF) ***(Sigtia et al., 2014)***. Needless to say, the latter results in a better model. The authors try an RNN-NADE as well which they use as their main model. More about this publication can be read under the subsection on the NADE.

  Short after the beginning of the recent era, several improvements to RNNs used music as a benchmark problem to be able to compare the performance of their addition to models with that of Boulanger-Lewandowski et al. ***(Boulanger-Lewandowski et al. (RNN-RBM, RNN-NADE), 2012)***. An early example evaluates mixed improvements to RNNs: gradient clipping, Nesterov momentum, the use of ReLU as an activation function as opposed to the sigmoid or tanh functions and the use of leaky integration ***(Bengio et al., 2012)***. The latter is a technique where long-term dependencies are taken care of by updating the hidden state of a unit with one fraction from the old state and one (the remaining) fraction from the output of the forward propagation rule. The improvements are also tried on the RNN-NADE ***(Boulanger-Lewandowski et al. (RNN-RBM, RNN-NADE), 2012)***.

  Another example evaluates, as part of the deep learning revolution, different ways to make RNNs deeper ***(Pascanu et al. (DT(S)-RNN, DOT(S)-RNN, sRNN)), 2013)***. The most straightforward way to achieve this is to stack several RNNs on top of each other with the model output coming from the top one. Each RNN has its own series of hidden states and forwards its output to the next layer RNN, effectively increasing depth and the capability of modelling complexity. However, there are several other ways to increase depth in RNNs and it turns out that extra layers can be inserted in the input layer, hidden layer or output layer. An extra layer in the input layer allows for deeper feature extraction from the input, whereas an extra layer in the output function allows for outputs changing more rapidly as a function of new hidden states. Extra layers in the transition between hidden layers allows for the hidden layer to change more dynamically, as a function of new input, and ultimately relates to the principle of highway networks. Finally, skip connections can be inserted to skip extra layers in the case where these mechanisms are needed only occasionally. Three final architectures are tried on music moddeling: the DT(S)-RNN, with deep hidden transitions as well as skip connections, the DOT(S)-RNN, with deep hidden to hidden and hidden to output transitions as well as skip connections, and the sRNN, which consists of stacked RNNs. The DOT(S)-RNN is tried both with regular units as well as maxout ***(Goodfellow et al., 2013)*** and Lp units ***(Gülçehre et al., 2013)*** in which case the latter performs the best. Maxout units are pooling units that generalize several regular non-linear functions used such as ReLU and leaky ReLU), and Lp units are further a generalization of maxout units.

Also, treating music as a benchmark problem, the notion of fast dropout ***(Wang and Manning, 2013)*** is ported to RNNs and subsequently tried on music ***(Bayer et al., 2013)***. The idea is that neural networks benefit from regular dropout in that they become more robust, however, it takes longer time to train since the dropped out units do not get their weights updated. This is remedied by the use of fast dropout where dropout is approximated during weight updates instead. This is further taken to recurrent levels and tried out on benchmark datasets.

When attempting to create and concretize the notion of unfulfilled expectations in music, something that the authors argue that even untrained listeners are very good at, an RNN, an LSTM and a so-called *clockwork RNN* (cwRNN) ***(Koutník et al., 2014)*** are trained on Bach chorales ***(O'Brien and Román (MusicNet), 2016)***. This is a subject that has been explored before and as always, it is hard to determine what the authors are after; in this publication, they conclude that perplexity, a measure related to the reciprocal of probability, increases when unexpected sequences of music occur. This, in turn, is very expected since the very definition of unexpected sequences in this context is a sequence that don't occur often in the training data, and thus the perplexity for such a sequence will be high since the probability of it will be low. It then seems like circular logic to inspect and argue about the phenomenon in question. What is interesting however is that the authors use a cwRNN which effectively models musical sequences better than both the RNN and the LSTM. The cwRNN has a partitioned hidden state where parts of the state are updated at different time step intervals, typically in terms of powers of two. This can be visualized in different ways and one can think of it like the hidden layer weight matrix is upper triangular which results in that only slower partitions have recurrent connections to faster but not vice versa. Hidden states are also only updated for partitions that should be updated at the given time step. The resulting behaviour is that the fastest partition updates at every time step, the second fastest updates every second time step (and then it only has recurrent connections from slower partitions and it does not change at all in the intermediate time step) and so on. The idea is that long-term structure should be preserved and available at all times, similar to how time-delay connections are used. The application is interesting and the authors argue that the cwRNN, as opposed to the RNN and LSTM, generates music that are consistent and in style. Unfortunately, no samples are available.

- **Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU):**

  As with RNNs, many recent architectures with LSTMs and GRUs only use the model to account for temporal context only, as opposed to also using its hidden state and one or more fully-connected layers to generate output ***(Lyu et al. (LSTM-RTRBM), 2015; Vohra et al. (LSTM-DBN), 2015)***. These models generally have some other architecture to generate the actual outputs, such as an RBM ***(Lyu et al. (LSTM-RTRBM), 2015)*** or a DBN ***(Vohra et al. (LSTM-DBN), 2015)***. Sometimes, LSTMs have also been used as a subcomponent of larger systems that model different subtasks with different architectures ***(Bretan et al., 2016; Mogren (C-RNN-GAN), 2016; Yu et al. (SeqGAN), 2016; Bretan et al., 2017; Chen et al. (FusionGAN), 2017; Guimaraes et al. (OR-GAN), 2017; Hadjeres et al. (GLSR-VAE), 2017; Hadjeres and Pachet (Deep-Bach), 2017; Hennig et al. (Classifying VAE, Classifying VAE+LSTM), 2017; Roberts et al. (MusicVAE), 2017; Roberts et al. (MusicVAE), 2018; Sabathé et al. (DRAW), 2017; Teng et al., 2017; Tikhonov and Yamshchikov (VRASH), 2017; Eppe et al., 2018; Koh et al. (C-RVAE), 2018; Simon et al. (MusicVAE), 2018)***. Finally as with RNNs, sometimes LSTMs are used as somewhat of a baseline when trying out more complicated architectures ***(O'Brien and Román (MusicNet), 2016; Yu et al. (SeqGAN), 2016)***.

  Beginning what is in this work referenced to as the middle era, Eck and Schmidhuber uses an LSTM for the first time in the history of music modelling with neural networks to model so-called twelve-bar blues ***(Eck and Schmidhuber, 2002)***. Their publication is thorough and clear and references both Todd and Mozer's CONCERT and evaluates different aspects in relation to these. In a first experiment, only a chord progression is modelled as a baseline and in the second, a pentatonic melody is modelled along with this chord progression which stays the same in all experiments. Two LSTM networks are used: one models chords only and the other melody but the latter gets fed the output of the former to account for melody. This is the first example of a publication where an online link to samples is available, alas not working at the time of this writing. It is, however, really simple to find a working website online holding the same content and samples.

  Also using LSTMs, both trying out the new architecture against older ones as well as testing a new representation of music, two toy problems and an attempt to model an AABA melodic sequence is undertaken ***(Franklin, 2004; Franklin and Locke, 2005; Franklin, 2006; Franklin, 2005)***. The new representation is inspired from Mozer's distributed representation of pitch as well as the chord representation used by Laden and Keefe, later refined by Mozer, and uses the notion of seven circles of thirds. With this representation, seven bits is needed to represent the pitches in an octave which is compared to the twelve bits needed both for a binary local solution as well as the distributed representation as laid forward by Mozer (height input removed for comparison so that Mozer's representation is only applicable for an octave). A similar improvement is shown in terms of chordal representation and even though the author(s) manage to more clearly prove that LSTMs are superior to earlier architectures, the major improvement with the data representation seems to be in terms of sparsity and learning speed. Later, a fine-grained representation for duration is added, and two separate networks modelling duration and pitch are used to model a written version as well as a live performed version (thus testing the duration representation on micro-timing) of the song Afro Blue ***(Franklin, 2006)***. Franklin argues that conditioning on chords and a first-beat-of-the-measure marker is more successful than when omitting the chordal structure input. The very same setup, now with interaction between the networks, in the form of the input of last step duration prediction to the next step pitch prediction and vice versa, is later used to learn several jazz songs ***(Franklin, 2005)***. At this point, plan units are incorporated as well and by inputting an unseen chord progression along with a plan, the network can generalize and compose new melodies after training.

When Eck takes on LSTMs the next time, he does it with LaPalme and with several new devices ***(Eck and LaPalme, 2006)***. First of all, the inputs, being last time step pitch and chord, have been augmented with data from corresponding places in earlier bars. Typically, material from two, four and eight bars earlier is supplied in what is called *time-delay connections*. The purpose of these is to simplify the processing of repetition, which the authors argue is a central component of music and, as such, also hard to learn. The input is at every time step given both to the LSTM network, for long-term dependencies, and to a feedforward network, for short-term dependencies. The output layer predicts the next chord and pitch and bases its predictions both on the output from the LSTM network and feedforward network. Eck and LaPalme train a vanilla LSTM and a feedforward network for baseline and show that their improved model performs better.

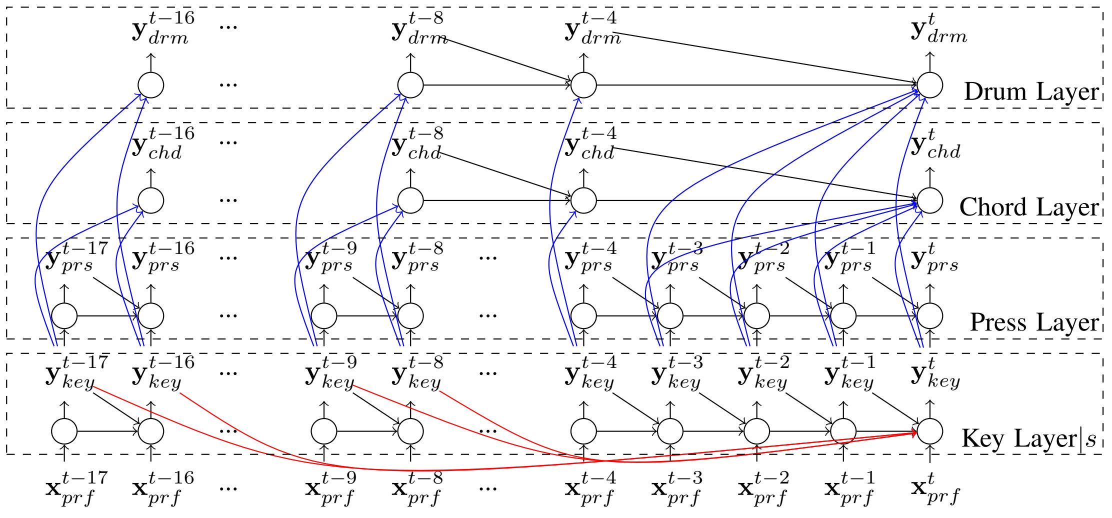
Shortly after the GRU surfaced, preliminary experiments indicated that it was hard to distinguish from the LSTM in terms of performance. Taking this concept to several modelling scenarios, among others music modelling, GRUs were tested against LSTMs, using an RNN with tanh as an activation function as baseline ***(Chung et al., 2014)***. The benchmark datasets from Boulanger-Lewandowski et al. were used ***(Boulanger-Lewandowski et al. (RNN-RBM, RNN-NADE), 2012)*** and the models were kept quite small to avoid overfitting which could otherwise ruin the comparison. Furthermore, the authors discuss the added functionality of the LSTM and the GRU, compared to the regular RNN, and give insightful tips as to why they have increased potential. After experiments, however, the authors conclude that even though they see hints of GRUs having the capability to beat LSTMs sometimes, they do not have enough evidence to suggest that this is the case in general. They manage to show, however, that the LSTM and the GRU is superior to the vanilla RNN, even though the latter surprises at times and performs remarkably well.

An early attempt with an LSTM only in the recent era has drawn inspiration from the trend-setting article by Boulanger-Lewandowski et al. ***(Boulanger-Lewandowski et al. (RNN-RBM, RNN-NADE), 2012)*** and uses a piano roll representation and the JSB chorales dataset ***(Liu and Ramakrishnan, 2014)***. The Rprop learning algorithm ***(Riedmiller and Braun, 1993)*** is used and the authors show its superiority, both in terms of learning speed as well as final accuracy, when compared to BPTT on the same problem. No preprocessing of input data is done and the architecture is quite simple and traditional, thus the novelty lies in the use of something traditional in the recent era as well as the use of Rprop. Also, the authors make an effort to use an evaluation metric which they however conclude do not correlate with the quality of the output music.

LSTMs and GRUs were tried against each other again in an attempt to model audio ***(Nayebi and Vitelli (GRUV), 2015)***. The model named GRUV uses spectrograms in the frequency domain instead of the actual audio in the time domain. In the former, amplitude of audio is a function of time whereas in the latter, it is a function of frequency in a signal which is a binned representation of some time interval. During preprocessing, N mono audio frames samples at 44.1 kHz are transformed into N real numbers and N imaginary numbers, effectively doubling the size of the input. The model then predicts the next spectrogram which can be transformed back into audio. Quite to the contrary of some results, the authors indicate that LSTMs perform slightly better in numbers, but much better when inspecting the results. The available sample sounds like distorted music ("musically plausible" according to the authors) whereas, apparently, the output from GRU models were more like white noise altogether. An interesting detail is that the acronym GRUV only appears once and its meaning is not explained.

Representing chords and drum patterns as text, linguistic models have also been used on music ***(Choi et al. (char-RNN, word-RNN), 2016)***. The authors use a char-RNN and a word-RNN (really LSTMs here) to model jazz chord progressions as text. The char-RNN models the text as characters and the word-RNN as words. The former has the advantage that only 39 characters are used whereas instead, the sequences are very long. The word-RNN, on the other hand, has 1259 different words but shorter sequences. A temperature is used during output to determine whether the models should be adventurous or play it safe during generation. The authors conclude that both of the models work well for this task. Later, drum patterns from Metallica are used as training data, using a 9-bit binary vector to signal activities across nine different drums of a drum kit. Here, only 4 characters are used (0, 1, space and a bar token) whereby a total of 512 possible words can be formed, however not all present in reality in the training data. This task results in very long sequences for the char-RNN which performs much worse than the word-RNN. Results sound good and musical.

Reviving some of the techniques from the early period, where a lot of implicit knowledge was encoded directly into the network, the authors behind the Song from PI publication uses a 4-tiered structure, where each tier is a 2-layer LSTM, to model pop music ***(Chu et al., 2016)***. Much statistical information from the training data is used to determine additional input annotations, such as a melodic profile and scale type. Time-delay connections are also provided in each tier from different previous time steps. The bottom tier generates the melody using most of the extra inputs provided, the output is then used as input to a press tier which determines the duration of the chosen melody note. The press layer simply counts and is reset to 1 whenever an articulation is made. The output of melody tier is also used, together with time-delay connections to determine a chord and an accompaniment for each time step as well. After training, the model can output music which is then postprocessed using statistical measures, heuristics and dynamic programming whereby the output chords and rhythmic patterns are reduced to one per half bar. The authors use a panel of evaluators to conclude that their music is preferred over music from the Google Magenta project. Some things are unclear from the article but the music sounds good to be automatically generated, however monotonous, similar and with a strong 2/4 feeling.



Graphic 3.2: Schematic over the 4-tiered Song from PI model taken from their publication. Each tier is a 2-layer LSTM and in the image, both the order of conditioning as well as the time-delay connections are illustrated.

In another attempt using GRU units with the reason that they, allegedly, have the same modelling power as LSTMs but are simpler, melodies are modelled using two networks: one for duration and one for pitch ***(Colombo et al., 2016)***. The GRU networks have three layers and each layer is fed the concatenation of the input of the entire network as well as the outputs of the previous layers. The pitch network also gets the prediction of the duration as inputs, which is not self-evident in any way. One can argue whether the use of a dissonant pitch on an emphasized beat signals that it should have a short duration or vice versa. The authors use their trained model both to complete an unfinished melody as well as generate melodies from scratch and argue that both results show long-term structure and that the model has an understanding of character, phrases and meter. No samples are available but the written music looks good.

The next attempt with language models uses a 2-layer word-RNN (really an LSTM) on two different word representations of music ***(Huang and Wu, 2016)***. The first representation uses the MIDI note messages as words with all tracks concatenated after one another whereas the second uses time slicing with the concatenation of the numeric representations of active pitches, confined to be three simultaneously, as tokens. Learnable embeddings are used as well and the authors show that structure arises in the latent space of these. The authors argue that their model accomplishes music of the same quality as an RNN-NADE ***(Boulanger-Lewandowski et al. (RNN-RBM, RNN-NADE), 2012)***, albeit without providing listening samples.

As part of the Magenta project, a Google Deepmind machine learning project for arts, RL Tuner is a model that uses an LSTM trained on melodies as a first step, and then uses reinforcement learning (RL) to improve the results ***(Jaques et al. (RL Tuner), 2016)***. The authors uses the expression note-RNNs, as an analogy to char-RNNs, to refer to RNNs that predict the next note in a melody, and train an LSTM on this task first. Then RL takes over and the trained LSTM is copied to three networks out of which one is held fixed and acts as part of a reward function. The idea is that the reward function is a balance between the predictions of this original network and general music theory rules, such that avoiding too much self-repetition, using reasonable size leaps, mostly using diatonic steps, starting and ending on the tonic note and more. The two other networks take part in so-called Deep Double Q-Learning and their weights are updated to form the final model. As a reminder, a deep Q network is a neural network that is used to approximate the Q function used in RL when its state and action space is exponentially large, and Deep Double Q-learning is a way of training such a network. The authors also attempt to use two modifications called G-learning and Ψ-learning and conclude through statistics, showing that the music theoretic rules are adhered to, that the attempt is successful. Samples sound reasonably good but are short. Later on, the same authors (with a few additions) present the same results but have now renamed the architecture to Sequence Tutor ***(Jaques et al. (Sequence Tutor), 2017)***. This publication is more general and elaborate and the authors, besides presenting the same results again, also try their model on other fields of application. A listener survey is added which indicates that the output of their three models is significantly preferred to the outcome of the note-RNN only. More particularly, the samples from the Q and Ψ models are preferred.

Another attempt at audio, inspired by GRUV ***(Nayebi and Vitelli (GRUV), 2015)***, tries to model spectrograms of music but uses different architectural improvements such as feature extractors, in the form of fully-connected and convolutional layers, and parallelization using GRUV as a baseline ***(Kalingeri and Grandhe, 2016)***. The authors try five different models and report improvements over GRUV for all. The authors then use a panel of 10 persons to evaluate their models and a so-called *bilinear model*, which has two parallell LSTM layers that add to the same output, and a model with a 2D-convolutional embedding layer, working on a series of time steps as a 2D map, are considered the best. The authors state that they expect the generated output of the architecture to sound "pleasing to the ear", albeit not necessarily like the input. However, no samples are supplied.

In Chord2Vec, inspired by word2vec and word encodings, the ensuing and preceding chords of an input chord are predicted ***(Madjiheurem et al. (Chord2Vec), 2016)***. Three models are tested and each have their way of first performing an embedding of the input chord, and then predicting the surrounding chords: the feed-forward network and NADE-inspired model uses a traditional embedding layer whereas the best performing model, a sequence to sequence LSTM, uses the final encoder state as embedding. The first two models use a binary traditional pitch encoding whereas the LSTM uses MIDI pitch numbers in lists that along with a special end symbol can give rise to lists with an arbitrary number of chords. The article leaves many questions unanswered and given that it focuses on embeddings, it is bizarre that the embeddings are only evaluated with respect to their performance in some datasets; it would be desirable to see some analysis on the latent space and if any structure is formed in it.

Also trying char-RNNs and word-RNNs, under the names char-rnn and folk-rnn, the ABC format is modelled in a larger context with headers and bar lines ***(Sturm et al. (char-rnn, folk-rnn), 2016)***. ABC files are preprocessed and simplified but still pose as a complete music format with both headers and the symbolic notation used by the ABC format to denote different durations and pitches. The char-rnn processes files character by character (and also models a format with more headers) whereas the folk-rnn tokenizes units of characters corresponding to specific pitches. Bar lines are included and even though word-RNNs usually have a much larger vocabulary than char-RNNs, the char-rnn and folk-rnn have vocabulary sizes of 135 and 137 respectively, something that probably contributes to the fact that neither model is reported as being superior to the other. LSTM units are used and both models are trained on folk music melodies after which they generate music that are in style with folk music and also show signs of structural patterns, even though there are syntactic errors in the format of the output. The authors also show how to use their models as an aid in a traditional compositional process; an interesting angle that assumes a more likely scenario, some would say, than that of neural networks composing music without any intervention of humans. Music produced, in part, by the model has also been played by live musicians and evaluated by composers, adding to this angle. The sample melodies sound musical and idiomatic of folk but wanders and lack goal bringing about a feeling of pointlessness.

Using a two-phase process, a model can be directed towards generating desired music using music rules (grammar) ***(Sun et al., 2016)***. The idea is that an LSTM is trained once in a regular fashion, after which the initial training set is expanded by inclusion of new training data that adheres to the musical rules learned. During this process, the trained model generates melodies, but every output not adhering to the rules is resampled. The resulting data is added to the training set and the model is trained anew after which the actual output of the model is produced. The authors use three different rules that forces output to be in key, not take too big leaps and belong to a triad (it is ambiguous what this means) and augment the training set by the use of these rules, both one and one and all together. It is then shown that the output music shows an improvement in terms of these measures when compared to the model trained after the first phase. This is, however, not very surprising since each training set has been expanded with around 50% of data that closely follows a given rule, thus it is elementary that the final network will show a higher degree of these features in the output.

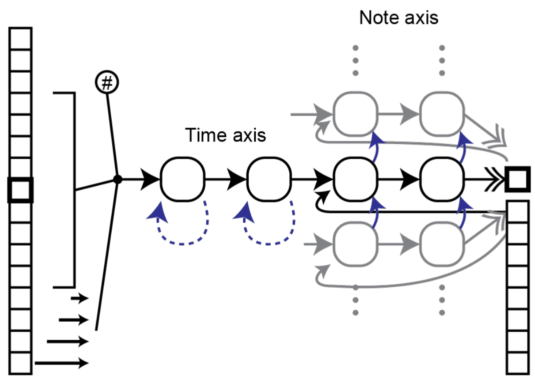
In an attempt to replicate the often referenced publication by Boulanger-Lewandowski et al. ***(Boulanger-Lewandowski et al. (RNN-RBM, RNN-NADE), 2012)***, Walder questions the conventional format of the piano roll combined with time slicing and lays forward a new representation along with (close to) state-of-the-art results on benchmark datasets as well as making these datasets available to the public, in versions preprocessed in a way more suitable to the new representation ***(Walder, 2016)***. The presented model uses a different angle and assumes that all rhythms are already predetermined. Furthermore, the model processes polyphonic music sequentially ordered in both time (from start to end) and in space (from low pitches to high), whereby only a single note is predicted at every time step. Extra inputs provide information about the total time in the piece (normalized to $[0, 1]$, the length of the note to predict as well as the offset of the note to predict to the current note. Time is measured in terms of quarter notes which, the author argues, allows for arbitrarily fine-grained durations. This is unclear however since the time values seem to be real-valued and the author does not address the problem of rounding. Inputs also indicate whether notes from the previous time step were turned off as well as whether the note at the last time step and the current take place at the same time (sequential polyphony). Some of these inputs seem to overlap in information and also add information to the prediction process that are usually not present in music modelling. The positive result of this sequential polyphonic representation is that the network does not model each simultaneous output note independently but instead, as in a NADE, based on all the lower notes starting at the same time as well. Through a heuristic improvement, only pitches above a certain lower bound (corresponding to the previous predicted note) are considered when modelling several notes simultaneously. The author argues that the model is an improvement over both how the NADE and the RBM are used, modelling sparse vectors of mostly zeros whereas this method only processes what is actually played. In the preprocessed training data, inspired by MIDI and more suitable to the proposed data representation, each song also has a unique plan and each note is associated with a track (instrument) as well for future expansion. The author claims that the addition of rhythmic information can easily be removed and instead modelled by the network, but it remains unanswered what the results would sound like then; thus this model is placed in a slightly other category than the bulk of music modelling networks and is thus hard to compare, in terms of samples, with these. The original model is temporarily altered for the author to be able to compare performance in a righteous way with previous results on benchmark datasets. This alteration is slightly unclear and because of this, the comparison remains uncertain. State-of-the-art results are close but the LSTM-DBN still performs better, even though these results may be optimistic, according to the author. When the model is used as intended, three categories of models are trained: models trained on each benchmark dataset, models trained on each benchmark data augmented by transposition to all keys and a model trained on all datasets. The sounding output available is from these models and it is thus hard to determine how the architecture does in free composition without predetermined rhythms. That being said, the available output sounds impressive in terms of pitch as well and the new data representation is a novel one worthy of further investigation. Results based on the new model are later published but it remains to be seen, however, if this article will be considered a milestone in the future and if the preprocessed datasets and modelling will contribute to a new benchmark for polyphonic music modelling.

Trying with both RNNs, GRUs, LSTMs and GANs, an LSTM sequence to sequence network turns out to be best at modelling text files, character by character, with melodies in an ABC-like format ***(Agarwala et al., 2017)***. Syntactic errors are occasionally committed but other than that, the output looks fine, which is confirmed by a panel that, to a noticeable degree, believes the cherry-picked output of the model to be human-made. Embeddings are used and a certain structure in the latent space is documented as well as rather long-term structure in the output as well as motivic use of seed content. Samples sound good and are mostly monophonic with the exception of a few duets that are acceptable.

In JamBot, two separate LSTM networks are used out of which one models chords, on a bar basis, that are later used to condition the other LSTM that models polyphonic music in a piano roll format ***(Brunner et al. (JamBot), 2017)***. For chords, a fixed vocabulary is used whereas in the polyphonic case, a standard piano roll format, not obliged to follow the current chord, is used. The current beat, current chord, next chord and the last time step piano roll column are used to predict the next. Polyphonic notes are sampled individually and a temperature parameter as well as a limit on the degree of polyphony are used to guide the generation process. The circle of fifths is derived from the latent space of chord embeddings and samples sound good but are made harder to evaluate because of the use of different electronic instruments as opposed to a regular piano sound. The authors claim that long-term structure is clearly present as well as consonant music. The former is not obvious from listening to samples.

Focusing on the transposition invariance property of music, the Tied Parallell LSTM-NADE (TP-LSTM-NADE) and Biaxial LSTM (BALSTM) draw inspiration from CNNs and form innovative models with good performance ***(Johnson (LSTM-NADE, TP-LSTM-NADE, BALSTM), 2017)***. The property of transposition invariance refers to the musical property of two identical pieces, one a transposed version of the other, often being considered the same piece of music (except for in very specific contexts). The corresponding property in images is handled by convolutions in CNNs whereby a feature recognizable by a certain filter will be recognizable no matter where in the input picture it is. The author accomplishes this in music by modelling all pitches using the same one pitch LSTM network that takes a window of surrounding pitch activities as well as a statistical measure over all active pitches in terms of pitch classes and also the MIDI number currently modelled.

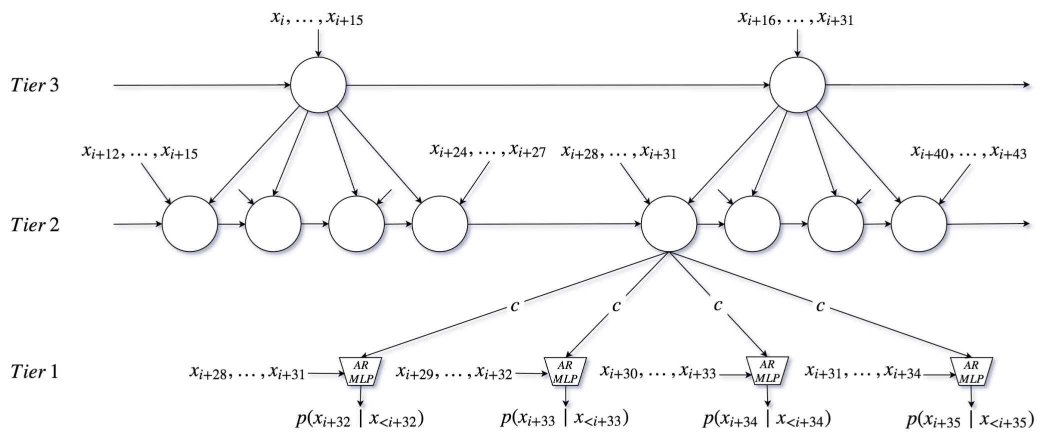
Since the same LSTM network is used for every pitch, it can be seen as feeding multiple LSTM networks with tied (same) weights in parallel. To solve the problem of independence among simultaneous pitches, the author uses both a NADE to model the output of a time step as well as a custom solution reminiscent of the previously described tied weight network. Three models are used: first an LSTM-NADE that takes as input the activations of all pitches at the same time, then a TP-LSTM-NADE where the NADE portion is modified to produce output based on two octaves of previous, already output, pitches at the same time step. Finally, a model with tied weights using the previous generated note of the same time step as well as the output of the previous part of the network for the current note. The first is assumed to be dependent on transposition whereas the others are assumed to be transposition invariant. The author tests these models on benchmark datasets and then attempts to compose with a modified BALSTM. The project seem to have begun as an online tutorial (`http://www.hexahedria.com/2015/08/03/composing-music-with-recurrent-neural-networks/`) which contains differences compared to the publication in terms of input and output data. Transposition invariance is indeed effectively demonstrated and very good results are acquired on benchmark datasets, however not state-of-the-art. As usual,long-term structure is suffering.



Graphic 3.3: Illustration showing the processing of a single pitch in the BALSTM, taken from the publication by Johnson. The two recurrent axes have two layers each.

Being innovative in nature, the BALSTM was later used in a three-part system designed to compose music, more specifically solos, in the style of famous jazz soloists ***(De Prisco et al., 2017)***. In this system, only the second part, the Predictor, is a neural network and its role is to predict the next note n-gram given the previous one. A note n-gram is a representation of n notes, and is a concept taken directly from natural language processing where n-grams are typically sequences of n words or characters. Each note is represented by integers indicating the root of the underlying chord, the quality (or type) of the underlying chord and the step number of the melody note relative the underlying chord scale. No duration is taken into account by the network. The authors use the value 24 for n and the first part, the Recognizer, contains SVMs that learn to distinguish a specific soloist. The Composer, the third part of the system, is a splicing system which has also been used in linguistics. Splicing systems are originally inspired from DNA and are defined by tuples of sets corresponding to an alphabet, an initial set of words and rules to form new words. In DNA, different subset sequences can be cut at special places and concatenated to form new sequences, thereby adding combinations to an existing set of sequences. In the proposed system, the Composer uses rules based on suggestions by the Predictor as well as more basic rules to determine how new music can be added to the initial repertoire consisting of the training data from a particular soloists. Durations are added via a heuristic process at this late stage. Samples sound very impressive and a panel of expert listeners vouch for both quality and stylistic coherence with dedicated soloists. However, it remains unanswered whether this technique can generalize to polyphony and entire genres of music.

Inspired by WaveNet ***(van den Oord et al. (WaveNet), 2016a)***, SampleRNN also models PCM-encoded audio samples, one at a time, but with a different strategy ***(Mehri et al. (SampleRNN), 2017)***. Instead of spanning a temporal context using dilated convolutions, multiple LSTM / GRU modules are used working at non-overlapping windows corresponding to different resolutions of the input. For example, the top module may take as input non-overlapping windows of 16 time steps at a time whereas the next layer does the same thing but with a window size of 8. After updating its internal hidden state, each module passes a projection thereof to the next module in a causal way (no module may be influenced by future time steps as a result of this). The lowest level module works at the sample level and is a feedforward network that takes the last few time steps into account only along with the conditioning vector from the higher module in the hierarchy. This processing is reminiscent of that of the clockwork RNN ***(Koutník et al., 2014)*** and the authors show great results, both in terms of results when compared to their own implementation of WaveNet as well as when a panel of listeners is confronted with generated results. Unlike WaveNet, it does not use external conditioning and so speech generated by the model sounds good and language-like but is plain gibberish upon closer inspection. In their results, a plain RNN (perhaps an LSTM or GRU) performs better than their implementation of WaveNet which makes the reader wonder whether their implementation is correct. Musical samples sound like music but less convincing than the output of WaveNet, despite the panel's preference of output generated by SampleRNN over output generated by WaveNet (as implemented by the authors).


$x_i, \dots, x_{i+15}$
$x_{i+16}, \dots, x_{i+31}$
Tier 3
$x_{i+12}, \dots, x_{i+15}$
$x_{i+24}, \dots, x_{i+27}$
$x_{i+28}, \dots, x_{i+31}$
$x_{i+40}, \dots, x_{i+43}$
Tier 2
c
c
c
c
Tier 1
$x_{i+28}, \dots, x_{i+31}$
AR MLP
$x_{i+29}, \dots, x_{i+32}$
AR MLP
$x_{i+30}, \dots, x_{i+33}$
AR MLP
$x_{i+31}, \dots, x_{i+34}$
AR MLP
$p(x_{i+32} \mid x_{<i+32})$
$p(x_{i+33} \mid x_{<i+33})$
$p(x_{i+34} \mid x_{<i+34})$
$p(x_{i+35} \mid x_{<i+35})$

Graphic 3.4: SampleRNN uses a hierarchical structure where each layer is responsible for a window of events. The bottom layer is a feedforward network. Illustration taken from the publication by Mehri et al..

Hadjeres, previously having showed interest in algorithmic composition where the user can guide and influence the generation of music, has also, this time together with Nielsen, arrived at a model which they call Anticipation-RNN ***(Hadjeres and Nielsen (Anticipation-RNN), 2017)***. This model has two parallell LSTM networks that go in different directions, much like in a bidirectional LSTM. However, the two LSTMs are called Token-RNN and Constraint-RNN, the former processing from left to right, the latter in the opposite direction, and they account for different things. The Token-RNN processes an input sequence with the purpose to predict the next step token based on what it has seen. However, at each time step, it is also conditioned on the output from the Constraint-RNN which takes as input a stream of constraints from right to left. The input constraint is set to the desired value, if the current token is to be fixed and to a special symbol if it is not. Binary masking is used to achieve this in reality. The idea is to model melodies where some of the notes in the melodies are fixed and the role of the model is then to supply samples of melodies where the constraints are respected but with, hopefully tasteful, additions in between. The model is trained on soprano melodies from Bach chorales and from an analytic point of view, the results look promising and the idea is inspiring. However, no sounding samples are available and there are fundamental parts of the model that are unaccounted for, for example how it is trained given this rather particular application of RNNs.

In BachBot, four-part chorales are modelled with harmonization and composition in mind ***(Liang et al. (BachBot), 2017)***. The publication is thorough with both novelties and good results, all from a transparent procedure. The authors model polyphonic music with time slicing but one note at a time in a sequential manner. Each token contains a pitch number and a boolean indicating whether the pitch is articulated or held. Fermatas are used as well as an explicit time step delimiter symbol. The architecture in itself is a standard LSTM network which, when trained, can both compose entirely on its own as well as with certain voices fixed. Thus, experiments show that depending on which voices to harmonize, and how many voices to complete, results vary. On a website (`http://www.bachbot.com`) a large group of people (2336 to the number at the time of this writing) with different levels of musical education were asked whether they could determine which ones of pairs of pieces that Bach composed himself. It turns out that when comparing Bach to full-fledged generations by BachBot, the results were not significantly deviating from random guessing, which is a very good result. It also turns out that it was easier to spot a genuine Bach piece when comparing to pieces were BachBot had only harmonized some voices. The authors argue that this might be because of the mix of styles that occur which might be inconsistent, and then goes on to analyze why and when different neurons in the network fire. This is a unique analysis of its type and even though it turns out to be non-conclusive, the activity of some neurons strongly correlate with particular musical events. Needless to say, the samples available from BachBot sounds great.

Inspired by a so-called Siamese Neural Network, StyleNet attempts to simultaneously model the dynamics of two genres of music ***(Malik and Ek (StyleNet), 2017)***. In a Siamese neural network, two sub-networks with shared weights take two different inputs and outputs a similarity measure used to, for example, verify a signature against a known original. In StyleNet, the principle is reversed and the input data, in this case music, is forwarded to two GenreNets, as the authors call it. A GenreNet is a bidirectional LSTM which, at each time step, outputs a vector of dynamics (or velocities as it is more often called in the context of music and computers) for whatever input vector it processed. The GenreNets do not share their weights and so the input music is, after initial processing in an LSTM layer, forwarded to both GenreNets where one recommends dynamics from a classical music point of view whereas the other one does the same but from the perspective of jazz. After training, a test panel cannot distinguish music interpreted by StyleNet from music played by a human being. The authors interpret that as something good, but at the same time, the panel can not distinguish the dynamic style of jazz from classical music. The samples indicate that dynamics are well predicted, however, it remains unclear why the authors choose to model two genres at a time; the number seems somewhat arbitrary and the GenreNets seem to benefit equally from training in sequence as two different networks instead.

In a hybrid architecture where an HMM provides chord changes and song part in a two-hot vector, an LSTM network then predicts the melody to go with it ***(Shin et al., 2017)***. The network models pop music and the HMM uses chord pairs, equivalent to two bars, as states whereas the emissions are one of four song parts typically found in pop music. The HMM is trained with statistics pulled from the training data and then provides the LSTM network with a two-hot vector. The LSTM then generates music as words, inspired by how recurrent networks can generate a sentence when given an image as input, made up of three components: pitch, duration and where in the current bar the pitch starts. The authors argue that the fact that they get rid of having to supply long notes as repetitions, which is necessary with time slicing, is beneficial and report that a large panel of test persons prefer their generated music over some reference models. During training they regularize the model by penalizing notes that are outside a given (rather narrow) interval. Samples have drums added and sound good but simple, which should be the case with pop music.

In a blog post, Simon and Oore demonstrates their PerformanceRNN while stating that the most important factors in music modelling are the dataset as well as the data representation used ***(Simon and Oore (PerformanceRNN), 2017)***. The actual architecture is of less significance, which is illustrated by how little space they use to explain what architecture they use. Alas, the fact that they use an LSTM network is made clear. Furthermore, the authors argue that micro-timing and dynamics are often neglected properties of music and choose a data representation with this in mind. Thus, the contribution of PerformanceRNN lies somewhere in between the symbolic domain and the signal domain (they don't only focus on the structural properties of music but also the live performance) and between composition and interpretation. However, as opposed to other works in the same domain (StyleNet for example), they focus both on composition and interpretation at the same time. In the data representation used, each represented pitch has both on and off events and time is progressed with a 100-dimension vector ranging from 10 ms. to 1 s. in 10 ms. increments, thus, the authors may progress by everything from 10 milliseconds to 1 second at a time. Finally, dynamics are also modelled in 32 dimensions. PerformanceRNN models one aspect at a time, sequentially, in a one-hot vector and uses temperature in the output layer to determine the amount of stochasticity in the output. Evident from the modelling aspects, the model is trained on MIDI datasets that are taken from live performances. Samples sound amazing, both from a compositional and a performance view, and it is hard to determine which one of these dimensions contributes most to the great results. Perhaps, the data representation used is beneficial from both viewpoints?

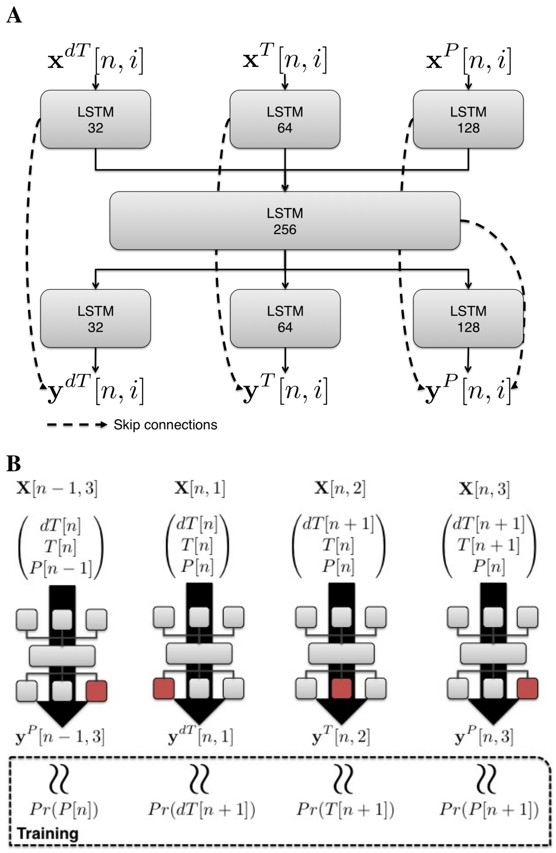
In Hierarchical RNN (HRNN), three parallel LSTM networks are used to model melodies operating on different time scales, much like a clockwork RNN ***(Koutník et al., 2014)*** but with the different scales in different networks ***(Wu et al. (HRNN), 2017)***. The highest hierarchical network models entire bars, more specifically bar profiles predicting the next one based on the current one. One level down, the beat network models beats in a similar way, now also using input from the bar network. During preprocessing, all bars and beats are classified, and subsequently represented by, one of 16 or 8 profiles respectively, which are also given to the lowest hierarchy LSTM network which models sequences of single pitches. The authors effectively show that performance increases by adding first the beat level network and even more so when adding both the beat and bar level networks. Lookback connections are also supplied in all three networks. A test panel indicates that the three-level network is preferred over the other two and that the output from it is perceived as music written by a human to a substantial level. The authors also show that their music is preferred over melodies generated by MusicVAE and MidiNet. Samples show some long-term structure in terms of rhythm, mostly. However, samples sound alike and there is not a big difference between the different models.

Deep Artificial Composer, DAC, is an updated version of a previous model by Colombo et al. and uses the same principle with two deep GRU networks that model duration and pitch, respectively, the latter conditioned on the former ***(Colombo et al. (DAC), 2017)***. The direction of conditioning is motivated by rhythm being easier, and more accurate, to predict, or conversely, pitch is more informative. The authors have added an extra layer in each of the networks, making them 4-layer networks, and have also changed the training process slightly and added support for MIDI to their work so that even though they use ABC as an input (which is converted to MIDI), the input of their model accepts MIDI making it more usable. Finally, the authors use a set of Klezmer melodies, aside from the Irish folk dataset used in their last publication, for training. They have developed their own evaluation measure and show that even though the generated music is different from both Klezmer and Irish folk, it resembles both, thus it has managed to generalize to a fusion of the two. With the evaluation measure used, the authors compare subsets of fixed sizes of generated material to same size subsets of the training data to see whether their model simply copies material or not. Temperature governs the existence of less likely melodic progressions and the authors are optimistic about their results. Samples sound either like Irish folk or Klezmer but show long-term structure and are impressive as monophonic results.

Ycart and Benetos focus on choices of parameters and preprocessing when modelling polyphonic music, using the most basic and simple LSTM network for the purpose ***(Ycart and Benetos, 2017)***. The authors then try out their model for an Automatic Music Transcription (AMT) task. Besides the more general choices of parameters, including the choice of learning rate, the number of hidden nodes in the state layers and more, a comparison is made between using a time-based and note-based sampling method. With the former, notes are sampled every 10 milliseconds from the input MIDI whereas in the latter case, the music is sampled using the notion of a quarter note according to which every sixteenth note is sampled. No matter the choice, a piano roll matrix is the result, constructed with two different sorts of time slicing in mind: the former related to the signal domain and the second related to the symbolic domain. The authors show that better results in terms of accuracy are acquired with a time-based sampling process but also argue that this is because more self-transitions are present in this data representation, thus, a model rewarding self-transitions will always score high, no matter the music. The note-based sampling process yields a representation where the model scores lower but instead, by inspection of next note probabilities, the authors show that the model has comprehension both of tonality and meter. When tried on the AMT task, the authors state that their models bring nothing to the table and the initial output, before postprocessing with a model, is often as good or better than the final output.

In DeepJ, a model that has a name inspired by the concept of a DJ (Disc Jockey), the BALSTM (Biaxial LSTM) is used with a few extensions to model polyphonic music from 23 composers ***(Mao et al. (DeepJ), 2018)***. As indicated as an extension in the original article, the authors add extra inputs to indicate rearticulation to be able to differentiate between long and rearticulated notes. The authors also include dynamics as an input, as well as composer or genre. More specifically, each piece is accompanied by a composer that is supplied from an embedding, via individual linear mappings, to each layer of the network and the idea is that the user should be able to interpolate between these with the trained model. Finally, the authors also use a convolution for the surrounding context notes, as opposed to feeding them directly into the network which was done in the original paper. A panel indicates that results are preferred over those of the BALSTM and that the music produced by DeepJ cannot be distinguished from original music from the same genre at a significant level. The authors show that the space of embeddings of composers has structure according to genre and suggest that the velocity might be good as a conditioning factor in determining what notes to play. Finally, it is indicated that DeepJ generates style-consistent music where baroque music is more contrapuntal, classical music is more homophonic and romantic music is more free in rhythm. Samples sound impressive and even though hints about different eras can be traced in samples, all music tend to be quite contrapuntal which, as a machine learning accomplishment, is great but perhaps not what the authors are looking for. Music from the classic era is sweet and less intense and romantic music uses a broader range and more dissonances. At `http://deepj.ai`, a showcase where the mix of different genres can be played around with, unfortunately with a mixed results.

With a new so-called normalized MIDI representation, BachProp uses three layers of LSTM networks to model polyphonic music sequentially by pitch, duration and offset from previous event, reminiscent of how MIDI itself works ***(Colombo and Gerstner (BachProp), 2018)***. The authors argue that much of the existing MIDI music is either recorded live or exported from MIDI writing software, all with different bad habits or peculiarities making the corpus of MIDI music available online a rather diverse one. In an attempt to unify it, a MIDI normalization routine is suggested which converts MIDI into a low dimensional but still exact and complete representation in three steps. First all activities from notes starting or ending are recorded so that every note is represented by both when it starts and ends (these are originally separate events in MIDI). After this, all values are normalized with respect to the length of a quarter note. Finally, the values are quantized to the closest of 21 fixed durations and expressive timing and malfunctioning software with inexact time stamps is a mere memory. BachProp models polyphonic music note-by-note where each event has an offset from the previous event, a duration and a pitch. The network itself has three layers with skip connection where the top and the bottom layer has individual networks for the three properties and a single large network in the middle. The idea is that each time step has three substeps where first offset is modelled, then duration is modelled also conditioned on the current time step offset and then pitch is modelled based on both current time step offset and duration as well as the context from before. This implies that some of the subnetworks stand still during two of the substeps which is not closer described. The authors use several less used datasets and a panel indicates that the output of BachProp is of the same quality as the music it was trained on. Generated pieces have also been played by live musicians with results that sounds great.

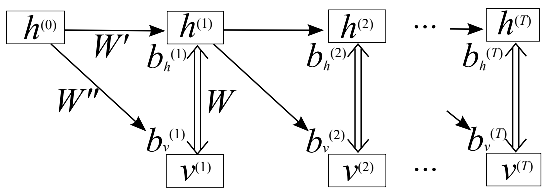


Graphic 3.5: BachProp using several LSTMs that are independently updated, the lower figure showing the mode of operation. Schematic taken from the publication by Colombo and Gerstner.

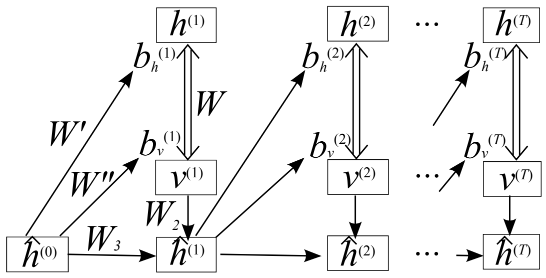
In MotifNet, the motive seems not to be to accomplish a model that can generate music superior to all other models, but rather investigate how to properly model the use of motives and the resulting self-similarity in music ***(Walder and Kim (MotifNet), 2018)***. Therefore, the authors focus on monophonic pitch sequences, without durations, only and demonstrate a very elaborate system with a plethora of subcomponents to handle the task. Their starting point lies in the notion of editing distance and they first lay out the theory after which they introduce their network which consists of mostly feedforward networks that model different functions used to determine the editing distance. A hyperparameter sets the maximum length of a motive (the authors use 5 when modelling music) and their model then searches through the previous sequence for similar motives in order to find the most likely continuation, effectively affecting the prediction. Several heuristic improvements are needed to do this without wasting too much time and the authors use both trees and priority queues to improve over the naive solution. An analogy function, also implemented in a neural network, designed to account for motivic similarities, but with an offset (for example the similarity between 2, 4, 7 and 10, 12, 15) is part of the system as well. In itself, MotifNet only has feedforward networks and a GRU network but an LSTM can be incorporated to account for better sequential context and the authors successfully show that the combination of MotifNet+LSTM performs best when compared to MotifNet only and a baseline LSTM. This model consists of many parts and it is very hard to keep track of how it works. For example, whether extra manual annotations are necessary or not is not stated and the training procedure, altogether, remains a black box. On top of this, neither written nor sounding samples are available making it even less convincing, even though the basis of the article addresses, and seems to partially solve, a very important and interesting topic that might be a key to future applications in music modelling. If the community is divided into two parts where one thinks that the model should be provided a minimum and solve all modelling and structures by itself and the other one believes that conditioning and complicated mechanisms (and perhaps interaction between different models and even then with guidance from humans) is the way to go, MotifNet is definitely a contribution in the latter category.

- **Restricted Boltzmann Machine (RBM):**

  One of earliest attempts to use RBMs for music is the Effective Boltzmann Machine (EBM) which is a regular RBM used as a sequence predictor where one context is modelled at a time. The size of the context can be varied but is often held at 3 or 5 steps in the experiments. At one time step, given a context of 3, the network is given three sets of inputs corresponding to a few octaves, where the used notes at a given time step are indicated. The task for the network is to predict the middle harmony (as a set of pitches) given the preceding and ensuing harmonies as well as the top voice (melody) of the entire context. After training, the user determines which of the input patterns are held fixed and by sampling from the model, a completion process occurs. By shifting patterns one step, one might then do the same thing again, an arbitrary number of times, to complete a longer sequence in terms of harmonies or melodies. By shifting the inputs, a chain of RBMs is formed which the authors call an EBM.



(a) RTRBM



(b) RNN-RBM

Graphic 3.6: Below is shown the RNN-RBM which is an improved variant of the RTRBM which is shown above. An RNN is used to output biases for an RBM whose output is then fed as next time step input to the RNN. Figure taken from the original publication.

With a milestone publication featuring RNN-RBM, a recurrent RBM to model polyphonic music, begins a new era in many aspects ***(Boulanger-Lewandowski et al. (RNN-RBM, RNN-NADE), 2012)***. First of all, the authors take a clear step away from the idea of the output function of recurrent neural networks modelling polyphonic output in a single time step, implying that all the pitches are independent from each other, and suggest that something much more powerful is needed. As such, inspired from the Recurrent Temporal RBM (RTRBM), they propose the use of an RBM at every time step of an RNN for this kind of modelling. The idea is that each time step contains an RBM, conditioned via the biases on the previous temporal context of the RNN, whose visible layer is the time step data of the music. This data is the input of the RNN which then uses its new state to initialize biases of the next conditional RBM, and so on. The publication if very thorough and multi-facetted and the model is first tried on some preliminary non-musical problems. The authors then use a battery of simpler models as well as an RTRBM, an RNN-RBM and an RNN-NADE (model similar to RNN-RBM but with a NADE as an output function which is attractive thanks to the possibility to calculate its exact gradients, something not possible with RBMs) and try them out on four big datasets that they subsequently make available, both raw and preprocessed, for future use. This publication sets the standard of end-to-end modelling with full range polyphonic data in each time step with nothing but the network and the music to use for training. Through its milestone character, the article also establishes the piano roll as a standard input representation and sets baselines on the used datasets for the years to come. They conclude that Hessian free optimization is beneficial for recurrent neural networks and that their RNN-RBM and RNN-NADE outperforms the competition with more or less margin. More remarkable is that the RNN-RBM actually outperforms the RNN-NADE in a series of ways and thus, the power of the RBM remains despite its rather heuristic and impractical training procedure. As a final undertaking, they show that using their models as priors for music transcription improves the results significantly.

Expanding, in a natural way, on the notion of making RBMs temporally aware, the LSTM-RTRBM uses an RTRBM but with some modifications that allow an LSTM to capture temporal context ***(Lyu et al. (LSTM-RTRBM), 2015)***. In contrast with the RNN-RBM, the temporal context is not entirely accounted for by the LSTM but also by the hidden state of the last RBM, thus making it a variant of the RTRBM rather than the RNN-RBM. The publication is short and it seems like the authors borrow results from other publications and use it as baseline without referencing it properly. Furthermore, they obtain state-of-the-art results on the datasets they use which makes it desirable to see a more conclusive and elaborate report. There are samples but not clearly linked to from the article. The samples sound better than random but are in the same style all of them and lack structure. The authors also claim that their generated music is more pleasant than the previous state of the art which seems to be an elusive remark.

The RBM as a model for music continues to draw interest and in convolutional RBM (C-RBM), the authors uses a 2D piano roll matrix as the visible layer ***(Lattner et al. (C-RBM), 2016)***. The piano roll has pitches on the vertical axis and time on the horizontal one and the hidden layer is divided into sets of hidden nodes that get their input from different filters that processes the visible layer. One filter is convoluted across the whole vertical space and a fixed time span and produces one hidden activation per position. A set of hidden nodes thus corresponds to the activations of the filter when convoluted across the visible layer and when using several filters, several such sets of hidden nodes get their activations. After training with persistent contrastive divergence (P-CD), the authors perform gradient descent on input noise, adjusting it while holding everything else fixed until it converges to a piece with desired properties. This process is called *constrained sampling* and the desired properties correspond to different aspects of music, resulting in a loss function on the input piece (noise). The desired properties need not be specified but are imitated from another input piece. The article is long and ambitious and several measures to evaluate the work are used. Samples from the models show deep musical quality even if the surface is sometimes a bit rough; without constrained sampling, the output sounds rhythmically convincing with structure but goes out of tune and shows no long-term structure. This is remedied partially by constrained sampling. The authors argue that the musical properties and how they are represented can be improved as well as the seams between parts of the output that seem very rough and without a musically convincing transition. According to a measure called information rate, the biggest improvement over baseline models comes with constrained sampling, and not from the inclusion of the convolutional aspects.

- **Deep Belief Network (DBN):**

  The first use of a Deep Belief Network, RBM-provisor, uses a 2- and 3-layer network with RBMs only, generating the melody for children's songs and jazz licks based on a 4-beat window of chords and melody on which the prediction of the next melody note is based ***(Bickerman et al. (RBM-provisor), 2010)***. RBM-provisor uses a fine-grained time slicing, and thus has a visible layer with more than 1000 units with which it is able to represent both duple and triple meter subdivisions of quarter and even eighth notes. The model is an attempt to prove that DBNs can be used to model music. The authors generate output melodies first according to the highest output probability which results in acceptable melodies but when instead sampling from the next note distribution, the melodies get very uneven and jumps of several octaves occur almost all the time which is very unidiomatic for music. RBM-provisor further outputs melody with duplet subdivision, despite having been trained with examples with triplet subdivision, and the authors conclude that they have shown that DBNs can be used for music modeling, but not much more than that.

Short after the milestone publication by Boulanger-Lewandowski et al. which effectively took us into what is here referred to as the recent era, a natural extension to their work was done in the RNN-DBN, replacing the RBM and NADE as output functions with a 3-layer DBN (with RBMs) instead ***(Goel et al. (RNN-DBN), 2014)***. The hidden state of the RNN from the previous time step is now used to initialize three biases for the conditional DBN: one for the visible layer and two for the hidden layers, as opposed to in the RNN-RBM where only two biases were needed (one for the visible layer and one for the single hidden layer). The publication is somewhat of a pure response to its predecessor and the same datasets are used. The authors compare their results to the ones presented by Boulanger-Lewandowski et al. and conclude that their work is on par with the state of the art (often slightly worse). Differences are that no preprocessing is done to the data and so it seems that the RNN-DBN is not trained on the actual piano rolls as made available by Boulanger-Lewandowski et al.. The authors use the lack of preprocessing as a reason for not beating the old state of the art, something which strikes the reader as a bad motivation since doing this preprocessing should not amount to much more time spent. The article lacks a lot of detail as well as samples but seems to be successful, at least in terms of evaluation measures.

DeepHear is a DBN with RBM layers that is trained on ragtime music ***(Sun (DeepHear), 2015)***. The author refers to it as an autoencoding DBN but since the autoencoder uses the same (but transposed) weights for transitions between corresponding layers in the (assumed) encoding and decoding part, it constitutes a DBN. After greedy training with contrastive divergence, it is fine-tuned using gradient descent and after that, sampling in the deepest layer results in four bars of output. A technique referred to as *prior sampling* is also used. Here, gradient descent is performed on the values of the deepest layer according to an error cost measuring the difference between a desired melody and the actual music at the visible layer. Once accomplished, the original melody, usually not clearly present in the result, is superimposed over the output. Samples sound impressive, mostly out of a harmonization perspective.

The authors behind the RNN-DBN later on improved their work by investigating the LSTM-DBN, now using four RBM layers instead of three ***(Vohra et al. (LSTM-DBN), 2015)***. The extension to use LSTMs instead of RNNs is quite straightforward and apart from this change and the added layer, the architecture does not offer anything new. The authors achieve new state-of-the-art results on benchmark datasets and claim that the generated music "possesses a richness that earlier models like the RNN-RBM did not display" without giving a shred of proof of this in the form of samples.

- **Sigmoid Belief Nets (SBN):**

  A sequential extension of the SBN is the temporal SBN (TSBN) which uses one SBN per time step ***(Gan et al. (TSBN), 2015)***. Instead of using the regular functions for updating the visible and the hidden layer, extra components from the last time step SBN are included so that the hidden layer uses both the last hidden layer and the last visible layer when initialized. The visible layer is then sampled from the hidden layer with additional information from the last time step visible layer. Several versions of this architecture are tried out and one such version does not take the last time step visible layer into account at all, neither for the hidden nor the visible layer. The authors argue that such a model is comparable to a Hidden Markov Model (HMM) with an exponential state space and call it a Hidden Markov SBN. In this setting, the output data of each time step is seen as an emission. The HMSBN is tried on polyphonic music data and furthermore, it is suggested that adding layers, which the authors also experiment with yielding the deep TSBN, does not add to the capacity. The TSBN (or rather HMSBN) does not improve on the performance of state-of-the-art models, even though they state that some of those measured numbers may be optimistic. Generated music sounds better than random and has some structure both in time and in space but lacks, as usual, long-term structure.

- **Neural Autoregressive Distribution Estimator (NADE):**

  In some publications listed under other architectures, a NADE is used as a subcomponent and less noteworthy in its context than the architecture under which the model in question is listed ***(Johnson (LSTM-NADE, TP-LSTM-NADE, BALSTM), 2017)***.

  In the milestone publication which started the era here referred to as the recent, a NADE was used at every time step of an RNN, conditioned on its hidden state to incorporate temporal context into the density modelling of each time step ***(Boulanger-Lewandowski et al. (RNN-RBM, RNN-NADE), 2012)***. The NADE was used as an alternative to an RBM, being more heuristic in nature, and gave the best results on several datasets when trained with Hessian free optimization (instead of gradient descent). More can be read about this publication under the subsection about RBMs.

  An RNN-NADE has also been used together with an Automatic Music Transcription (AMT) model ***(Sigtia et al., 2014)***. The latter is a model that transcribes audio into sheet music and the idea is to improve its accuracy by using a music model to impose constraints and guide the transcription. This, in turn, requires a tractable probability density function which an RBM, despite its effectiveness, does not offer (due to the partition function being infeasible to calculate for larger than very small network sizes). Furthermore, the authors agree with Boulanger-Lewandowski et al. and argue about how RNNs with polyphonic output model output as a multi-dimensional variable where each pitch is independent of all others;

something that is utterly wrong in the context of music. The idea is therefore to use the RNN-NADE which offers this tractable probability density as well as output at every time step where each variable is dependent of the others (according to some ordering). The authors also test a plain RNN, as a sort of baseline, as well and train both models both using stochastic gradient descent (SGD) and Hessian free optimization (HF) on the unprocessed Nottingham dataset of folk music using 200 time steps of piano roll to predict the next (single) time step. Three configurations are then tried: the first with the AMT model only, the second with the AMT model followed by using the music model after which the transcription is output with both of these in mind, and the third which is the same as the second but instead, the AMT model is run again with the music model as a prior. The results are not surprising and the RNN-NADE is better than the RNN, HF is better than SGD and the configurations improve the output transcription by every added step, thus the third configuration with the RNN-NADE trained with HF performs the best (all configurations improve over baseline significantly). The music to transcribe is a live Bach dataset and when trying to use a music model trained on a Bach dataset instead, the authors get results that are worse than with the Nottingham dataset. The argument is that folk music is more general and Bach music is, in itself, harder to predict, thus, using a more general set as vague guidance is better than using a model trained on the rather specific music by Bach.

In another contribution from Google Brain in a joint effort with University of Montreal, both prominent actors on the machine learning scene, CocoNet is described which is an orderless NADE realized with a CNN modelling the individual conditionals ***(Huang et al. (CocoNet), 2017)***. Coco stands for Counterpoint by Convolution and four-voice Bach chorales are modelled with four parallel piano rolls. In an orderless NADE, the order of the variables are made unimportant by generalizing over all possible orderings. This sounds like an expensive task but thanks to certain properties, this can be done efficiently ***(Uria et al., 2014)***. Subsets of the input are masked out and the network learns to restore these from the given context. When generating music, the user can either use ancestral sampling, as used normally in the NADE, whereby each ungiven variable is sampled one at a time. The authors however are better off using blocked Gibbs sampling where all the missing variables are sampled at one time (as though they were independent). This process is repeated a number of times and in each iteration, the number of masked out notes are decremented (annealed), resulting in something more similar to plain Gibbs sampling early on and ancestral sampling towards the end. Needless to say, this approach can be used to complete partial scores or generate new music and according to the authors, it is a strength that the network is given the chance to correct its own mistake through the process of repeated sampling. A panel of listeners indicate that with Gibbs block sampling, the generated samples cannot be distinguished from real Bach at a significant level. The opposite applies to the difference between samples generated with ancestral sampling and Gibbs block sampling. Generated samples sound indeed very good and the only thing missing in this article is a more detailed description of how the proposed model actually can be seen as an orderless NADE.

- **Autoencoder (AE):**

  The RAAM (Recursive Auto-Associative Memory) is a sort of autoencoder that decreases the size of the input by recursively encoding pairs of inputs, according to some rule, until no pairs remain after which an inverted decoding process takes place ***(Large et al., 1995)***. This network was tried on melodies in order to determine whether a reductionist view, more specifically the one offered by Lerdahl and Jackendoff, is sensible. The authors train the network one some melodies and then test its generalization capabilities by exposing it to variations of the previous melodies and altogether new melodies. Results indicate that reductionist theories indeed are plausible and that such latent structural material can be learned by a neural network.

  In the recent period, DeepAutoController uses a deep architecture with two stacked autoencoders to model spectrograms transformed from an audio signal with Fast Fourier Transform (FFT) ***(Sarroff and Casey (DeepAutoController), 2014)***. The authors are not very optimistic about their results but test their architecture methodically. Training is done greedily, layer by layer, after which fine tuning of the entire system is done. They also act exemplary by documenting what software frameworks were used as well as list links to both samples and source code. While training, Gaussian noise is added to make the model more robust and a series of sizes on the different layers as well as percentages of noise are tried out. Finally, a program to use with the model is devised in which all sorts of manipulation of the hidden units, where the latent representation is learned and subsequently from where new compositions start, can be done.

  In one ambitious attempt to model melody using three networks for individual subtasks, an autoencoder is used to encode musical units as a starting point ***(Bretan et al., 2016)***. A musical unit is typically anything from one to a few measures and the input to the autoencoder is a feature vector of a single musical unit represented in a bag-of-words (BOW) manner. The BOW model uses word n-gram counts to represent sentences which are then not exactly represented since the overall order is not accounted for. Taking this concept to music, the BOW model takes duration n-grams, pitch n-grams and a lot of statistical data based on the pitch and duration properties as a representation of a musical unit. The authors are inspired by both recombinacy, as a means to compose as indicated by Cope ***(Nierhaus, 2009)***, and the Text-to-Speech (TTS) field, where combinations of units are formed with good results on the task to form longer sentences. The autoencoder is trained using a loss that is based on cosine similarity. The input data is augmented by both transposition and rhythmic alterations after which a total of 80 million units are acquired. To reconstruct a measure from a BOW representation, cosine similarity is measured against the musical units of the database after which the one with the largest similarity is picked. At this point, the authors can input a melody and then output a variation of it according to their model; a task that produces good results. Linear interpolation between units can also be accomplished in a satisfactory way. Expanding the model, the authors now use a feedforward network as a Deep

Structured Semantic Model (DSSM); a deep network that creates an embedding of its input with the purpose of aligning such inputs that should be considered similar according to some measure. The same BOW model as earlier is used with an even smaller latent representation as output. The idea is that musical units is summarized and the DSSM is trained by a loss based on cosine similarity to map consecutive units to similar vectors in the latent space. After training, the DSSM is used to measure which of a set of candidate units (given in BOW representation), given a first, that is most appropriate to use as a continuation. This, along with an LSTM that, given a few units of music to determine the most likely ensuing pitch, is used to accomplish a composition model that samples from the latent space of the autoencoder and uses the LSTM and DSSM to determine which units should be attached to an emerging composition that begins with a seed. Statistical results leave much room for improvements but listening samples are impressive but monophonic.

Again using autoencoders, Bretan, along with colleagues, attempts to create latent representations of bars of music ***(Bretan et al., 2017)***. Latent representations are created using several different models which uses convolutional layers before flattening to fully connected layers yielding a 100-dimensional latent representation of each bar. Denoising autoencoders are tried (with several different kind of denoising properties) as well as predicting and context autoencoders. At one point, only an encoder is used and the output is mapped to a categorical output for each of the composers that contributed to the training data. After creating a database of latent representations, two tasks are tried out. First, in the prediction task, an LSTM is given seven consecutive bars, as latent representations, and are to predict the eighth. Secondly, the composer of each latent representation is to be determined. Not surprisingly, the latent representations created by a context autoencoder, designed to predict the surrounding bars (in additive form) based on an input bar, and a latent representation created specifically with composers in mind, perform best.

Eppe et al. wish to revisit the subject of modelling raw audio, but with very simple means using only an autoencoder with GRU networks as encoder and decoder ***(Eppe et al., 2018)***. The modelling is done in the Mel spectrogram domain and the authors use a large MIDI dataset, convert it into synthesized audio files and then use these as input. The authors here model single instruments individually, as a proof of concept, but a thought scenario for their model, once perfected, is to form part of a music ensemble where the input is streams from the other instruments whereby the model can generate a missing instrument. It is suggested that their simple model performs well in terms of timing whereas as far as pitch and variation are concerned, the results are slightly less optimistic. Different models are trained for each of three instruments and when generated (reconstructed) outputs are concatenated to form multi-track samples, they sound almost too good to be true. To summarize, with a simple network, the authors manage to get seemingly better results than what is acquired with far more advanced architectures in, for example, WaveNet or SampleRNN which is remarkable. This fact is not specifically addressed and one conclusion is that the use of Mel spectra is optimal for this task. Another suggestion is that real audio generated synthetically from

MIDI files is far easier to model than non-synthetic audio, which has been modelled with less impressive results earlier.

- **Variational Autoencoder (VAE):**

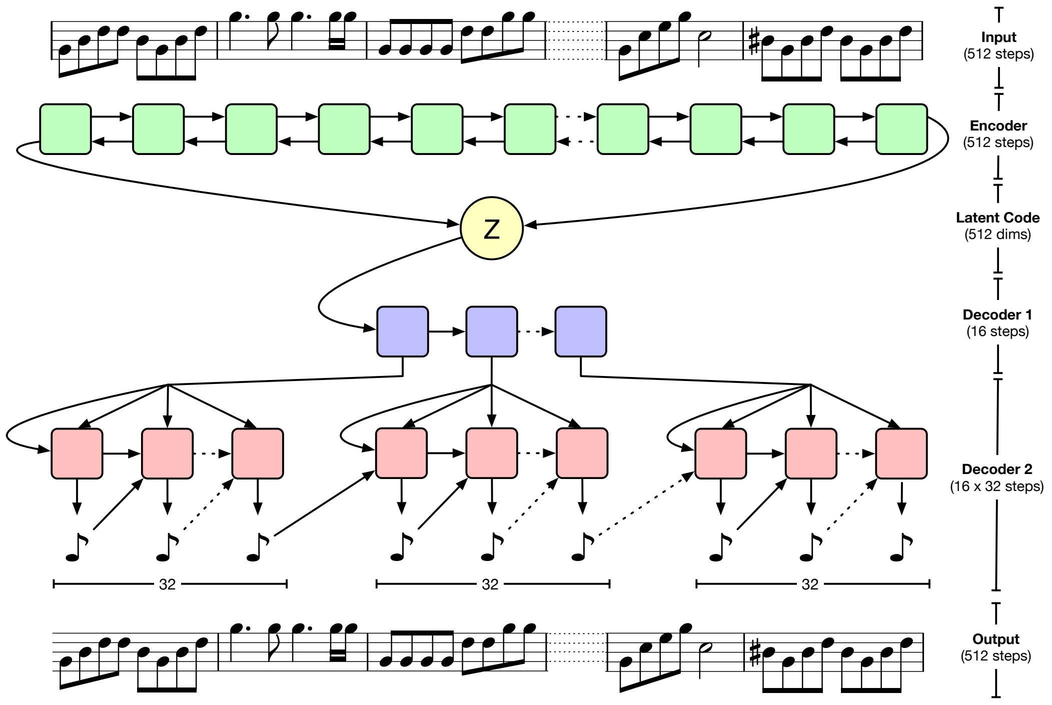
  One of the first attempts to use a VAE in recurrent networks to model music is the Stochastic Recurrent Networks (STORN) which maintains two parallel RNNs, called recognition and generating models, with a VAE in between at every time step ***(Bayer and Osendorfer (STORN), 2015)***. The idea is to incorporate temporal context into the latent space of the VAE, as well as stochasticity in the hidden state of the generating RNN. The former is accomplished by the recognition model that, at every time step consumes the input and outputs parameter for the latent space. The generating RNN then takes two inputs as opposed to the standard one: the same time step input, as did the recognition model, and a sample from the latent space of the VAE. The output of the generating model is then used to predict the next time step input. With this setup, both stochasticity in the hidden state of the generating model, via inclusion of data from the latent space, as well as temporal context in the latent space, via the recognition RNN, is accomplished. The authors report good results on benchmark datasets when compared to models in the same class, albeit architectures that don't assume independence among variables in the output vector of each time step are still better.

  Another early attempt, VRAE, uses the VAE in an entirely different way in a sequence to sequence (seq2seq) manner instead ***(Fabius and van Amersfoort (VRAE), 2015)***. Thus, no temporal context is incorporated in the VAE and only the final hidden state of the encoding RNN is used to parameterize the distribution of the VAE. After sampling from this distribution, a decoding RNN reconstructs the input with its initial state initialized by the VAE. The authors use very little training data and argue that the results might be improved by the use of LSTMs instead of RNNs and give a few other hints on what might be improved. The strength of this publication is that they show how a VAE might be used in a recurrent model generally, and for music specifically. The medley published by the authors sound like chunks from the training data and so questions about the generalization capabilities are left somewhat unanswered.

  By adding an extra regularization term to the VAE loss, one could induce a latent space with even more structure than usual ***(Hadjeres et al. (GLSR-VAE), 2017)***. The model is called Geodesic Latent Space Regularization VAE (GLSR-VAE) and the idea is that so-called attribute functions, that measure ordered quantities in the output reconstructions, are used to control the used latent representations and the authors effectively show that their method generates a latent space which, in regard to a certain dimension, yields an output with an increasing quantity as the value of the dimension increases. The quantity could be for example the number of output notes in a bar. The model uses LSTMs in a sequence to sequence fashion and only the final state of the LSTM is used to parameterize the latent space, which is assumed to have a standard Gaussian prior. In the output phase, the sample from the latent space is used only to initialize the decoding LSTM. The concept in this article is very intriguing and interesting which makes it even more regrettable that it lacks some detail as well as listening samples.

Investigating how generated polyphonic music can stay in key better, the Classifying VAE and Classifying VAE+LSTM are proposed ***(Hennig et al. (Classifying VAE, Classifying VAE+LSTM), 2017)***. The problem is that even though the latent space of a VAE is structured, how do we constrain what we sample from it? Starting from a Conditional VAE (CVAE) ***(Sohn et al., 2015)***, where extra annotations are used to condition the latent space and achieve this constraint, the authors argue that a similar result, but without manual annotations, is desirable. They then use a feedforward network to model the mean and variance of key classes to which a certain input belongs. This is done once per sequence and during training, the Krumhansl-Schmuckler algorithm is used to infer the key of a given sequence, used for training the feedforward network. After this classification, the main model takes as input both a piece of data and the sampled key to form the parameters of a latent vector. After sampling, data is decoded from the latent space sample along with the key as well as additional data. The main model is both implemented as a VAE and as a VAE+LSTM (hence the names) and in the first case, data from one time step is reconstructed given the last time step data, key and a latent space vector whereas in the second case, an LSTM network (more specifically the STORN network ***(Bayer and Osendorfer (STORN), 2015)*** is used whereby a whole sequence is to be reconstructed at once. Surprisingly, the recurrent and the feedforward models perform similarly in many aspects. Classifying VAE and Classifying VAE+LSTM generates music that stay in key better than corresponding models without conditioning on key, even though the models do not achieve remarkable results on benchmark datasets. The main achievement lies in the automatic key detection and the incorporation of this mechanism into the network. Samples sound okay and stay in key without sounding remarkably well.

A noticeable model from the Magenta community, a subproject of Google Brain focused on creative applications for machine learning art, that several articles have been based on is MusicVAE; an architecture with only LSTMs and a VAE ***(Roberts et al. (MusicVAE), 2017)***. The authors train on three types of data: 2-bar melodies, 32-bar melodies and 32-bar trios with lead, bass and drums and their architecture builds on the sequence to sequence paradigm. Using a bidirectional LSTM as encoder and then a deep LSTM as decoder, the MusicVAE reconstructs 2-bar melodies with grace. For the remaining experiments, the architecture is slightly altered and a hierarchical property is added where a first LSTM decoder outputs 16 pieces of latent code of which each is decoded as 2 bars by a lower hierarchy (deep) LSTM network. In the case of trios, the lowest hierarchy LSTMs are in reality one LSTM network per instrument. In each low hierarchy model, the input latent chunk is given as input to each time step as well as contributing by being the initial state of the LSTM. The authors also use scheduled sampling, as opposed to teacher forcing, and show that this increases the model's dependency on the latent code and improves the result. The samples sound impressive and a remarkable interpolation between two 2-bar melodies are showed as an example of what the MusicVAE can do. For this astounding example, spherical interpolation between two points in the latent space was used. One of the limitations of the model is that it cannot generate music in an unlimited way. In a more thorough, follow-up article by an extended crew, the authors show that spherical interpolation with a hierarchical decoder, over linear interpolation and spherical interpolation in a flat decoder, result in much more likely samples, in terms of the model's own loss function ***(Roberts et al. (MusicVAE), 2018)***. Furthermore, teacher forcing is now used instead of scheduled sampling and it seems like the improvements acquired from using the latter is now expected from the model itself, as opposed to being dependent on the type of training input enforced. The high-hierarchy step of the hierarchical decoder is now called the conductor and the authors discuss latent vectors and so-called *attribute vectors*, acquired by averaging latent vectors coming from all samples with a typical property, thoroughly. Finally, a panel of test persons indicate that when determining which of two samples are more musical, the output of a hierarchical decoder cannot be distinguished from real music at a significant level whereas with a flat decoder, both output from a hierarchical decoder and real music are significantly more appreciated.



Graphic 3.7: Illustration of MusicVAE with a hierarchical decoder taken from the first publication by Roberts et al..

Using a model called DRAW ***(Gregor et al., 2015)***, reminiscent of STORN, Beethoven sonatas are modelled from a very small training set of 37 MIDIs ***(Sabathé et al. (DRAW), 2017)***. The authors devise a 17-dimension vector representation of music pieces that is based on statistics about the quantity and quality of notes and uses the Malahanobis distance to determine similarity between two pieces, or between a piece and a set of pieces represented by the means of the vectors of its constituent pieces. The DRAW architecture has two parallel LSTMs with a VAE at every time step in between, where the decoding LSTM iteratively works on its output for a predetermined number of times. The original DRAW can use attention for the model to focus on specific parts of the input, however, this feature is omitted by Sabathé et al. since it did not improve the results. The authors show that, according to their measure, the output of the trained model generates music that is more similar to music by Beethoven than to random music or music from another genre. As always with measures of this kind, one might ask oneself if the captured properties are essential or not. In this case this is done by, after determining that according to some measure the output music should be similar to music by Beethoven, listening to the music and verifying this. In this case, the music is only 13 beats long and sounds okay, but unstructured and does not remind of Beethoven very much.

Using GRUs in a sequence to sequence network with a VAE modelling melody, the underlying chord structure is available as conditioning throughout ***(Teng et al., 2017)***. The architecture is one of the deeper of its kind and 12 layers are stacked both for encoder and decoder. The authors encode 8-bar segments and go through in detail how they extracted melody and chord structure from the training set. A relative encoding is employed so that the authors don't have to worry about key at all during training. Skip connections are used to give the network the option to not use all of its layers, if needed, and even though it is unclear exactly how, the chord structure is given as conditioning at all time steps, both in the encoder and decoder, at up to two chords per bar. It is the idea of the authors that this will lead the model to not encode the chordal structure in the latent space since it is available to the model anyway during decoding. This turns out to be true, which is shown by letting the trained model decoder start with the same latent sample but provide it with different chordal structures after which it effectively adapts the generated melody after the new chords. The authors also show vague implications of structure in terms of genres in the latent space and suggest that their music shows long-term structure and motives. Unfortunately, evaluation of all kinds are missing and only some examples are shown. During generation, a grammar is used to generate a chordal structure and a form which is then used by the model to generate melody. To generate similar parts, the authors add Gaussian noise to the initially sampled latent code. Alas, it is unfortunate that so little is described about this grammar. Samples sound great but monotonous, which is the case with pop music anyway.

Using the VAE in a sequence to sequence setting with some recurrent network model as encoder and decoder becomes more and more common in the later half of the 2010's and a thorough analysis using this architecture for language modelling was done in 2015 ***(Bowman et al., 2015)***. In this article, the authors state that a powerful decoder, such as an LSTM network, can entirely ignore the latent code and still get good results in terms of the loss function. However, such an aspect will not show until generation whereby the model will not be as influenced by the sampled latent code as desirable. The authors show different ways to handle this, one being weakening the decoder by not supplying already generated tokens or by randomly replacing already generated words with an unknown token (the authors call this word dropout) effectively weakening the decoder. It is also discussed how adding highway networks ***(Zilly et al., 2016)***, essentially making the hidden to hidden transition deeper and offering optional residual connections in a recurrent network, improves the results on language problems. Inspiration from these models is drawn in the Variational Autoencoder Supported by History (VRASH) where monophonic music is modelled ***(Tikhonov and Yamshchikov (VRASH), 2017)***. The authors include MIDI meta data as input to all time steps during both encoding and decoding and, additionally, add noise to the sampled latent vector before decoding. They also supply already generated tokens as input to the next generation time step, hence the alteration justifying the "supported by history" part. The VRASH performs only marginally better than competitors but manages long-term structure and the production of interesting melodies better. The article lacks a lot of information, for example why noise is added to the sampled latent vector as well as the depth of the highway layers used. Furthermore, no evaluation of the inclusion of MIDI meta data is declared. Generated music sounds monotonous but definitely has some pregnancy to it, especially compared to the simplest baseline model. Both the VAE and the VRASH models compose nice melodies, but it remains unclear, what the difference between these models are, except for the noise added to the sampled latent vector in VRASH.

When modelling music similar to an input, the Convolutional-Recurrent VAE (C-RVAE) uses the same latent encoding of the input, used for encoding when decoding in a sequence to sequence network with a VAE modelling the latent space ***(Koh et al. (C-RVAE), 2018)***. The idea is that a CNN first creates a sequence of latent representations for polyphonic input frames in piano roll notation, typically half a bar each, and supply these, in order, to the GRU encoder while encoding. The output of the encoder is then used to parameterize the distribution of a VAE from which is sampled a latent vector that initializes a GRU decoder which, in turn, is also fed the latent representations used by the decoder (possibly projected onto a different dimension). At each time step, a generated frame is output and after the model is trained, one simply samples a latent vector and then supplies a latent representation of an input, as a conditioning, to which one wants to generate a similar piece of music. The authors use this idea because their final goal is automatic music generation for video games where one might want to generate music from an environment similar to another but yet want new music. It is an ambitious attempt but they fail to make samples available that show how they have succeeded with their task; samples sound either almost exactly like the input or not at all and it would have been interesting to get multiple examples of what was generated from different latent vectors using the same conditioning piece.

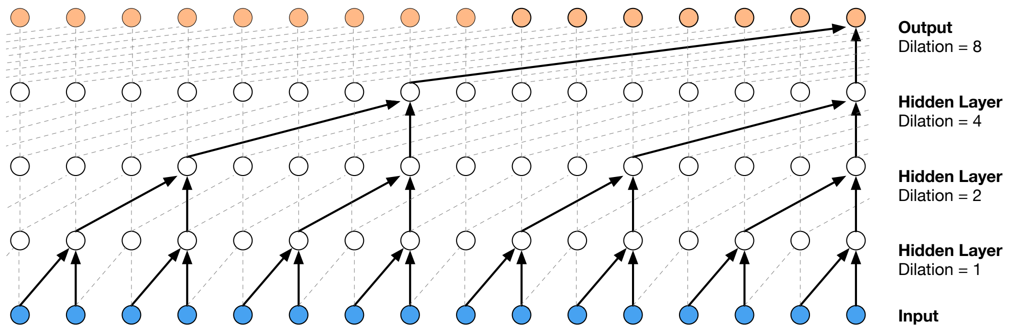
Building on the work done with MusicVAE, but instead using a data representation inspired from PerformanceRNN, 4-beat measures for multi-track (polyphonic or monophonic depending on the individual instrument) music with 2-8 instruments are modelled individually ***(Simon et al. (MusicVAE), 2018)***. The on and off messages with additional inputs to move time forward, now in fractions of a note value as opposed to time as was used in

PerformanceRNN, are used along with velocity modifying inputs for each of the input instruments. Input also contains information about which instrument the track is for, which also includes drums, and encoding is done with bidirectional LSTMs in two steps: first each track is processed forming one latent code per track after which a second bidirectional LSTM processes this sequence to generate the parameters for the latent space of the VAE. After sampling, a decoder conductor, as it is called in previous papers with MusicVAE, outputs a number of latent representations which are then forwarded to unidirectional LSTMs that generate tracks from each of the latent codes. Chord conditioning takes place both in every step of the encoder as well as the decoder and the authors successfully show that the model manages to output reasonable music adapting itself to the output chord conditioning when using the same sampled latent code in multiple attempts. Furthermore, interpolations and latent space vector arithmetic are shown and even though samples are short, manipulations are impressive and indicative of what the future has to offer.

- **Convolutional Neural Network (CNN):**

  Convolutional layers can be seen as feature extractors with specific properties and can subsequently be used as such on any input, including music, where its contribution may amount to that of a subcomponent ***(Kalingeri and Grandhe, 2016; Yu et al. (SeqGAN), 2016; Bretan et al., 2017; Chen et al. (FusionGAN), 2017; Dong et al. (MuseGAN), 2017; Guimaraes et al. (ORGAN), 2017; Huang et al. (CocoNet), 2017; Yang et al. (MidiNet), 2017; Koh et al. (C-RVAE), 2018; Mao et al. (DeepJ), 2018)***.

  In a inspiring attempt from 2016, a team from Google Deepmind announced WaveNet which models speech and music based on the raw wave form of audio modelled as a 16 kHz stream of 8-bit integers downsampled from 16 bits using PCM encoding ***(van den Oord et al. (WaveNet), 2016a)***. The architecture builds on PixelRNN, and more specifically on PixelCNN ***(van den Oord et al. (PixelRNN, PixelCNN), 2016b)***, and uses several modern techniques along with a 1-dimensional convolution over a large, but bounded, receptive field of the history to predict the next audio sample. More specifically, the terms causal and dilated convolutions are used, out of which the former simply refers to a convolution that only uses input from a history, according to some ordering, to predict the next token (that is, no peeking ahead). In a speech and music context, what is seen as history is quite straightforward, as opposed to in an image, where an ordering must be artificially induced over the pixels. Furthermore, a dilated convolution is a convolution where the filter inputs are spread out according to some number. A simple example with a dilation of 2 skips every other input and thus doubles the receptive window of the filter but halves the resolution of the filter. It turns out that by stacking dilated convolutions, where the dilation increases exponentially, the receptive field of a bounded history increases exponentially as well, while the number of parameters increases only linearly. This is in contrast with non-dilated convolutions where a stack grows linearly both in receptive field and in the number of parameters. Furthermore, skip connections and residual connections are also used as well as 1 by 1 convolutions to reduce channel depth. Finally, gating (where a parameterized portion of some input is forwarded in line with how an LSTM works) is also employed in each processing layer. By bounding the history, the network can be parallellized which is not the case when working with infinite memory RNNs, which in reality are bounded as well since they have a problem modelling long-term structure. WaveNet produces mostly gibberish, speech-like sound when run without conditioning, but reach remarkable results when conditioned on voice and content, which can be done in several ways. When generating piano, the receptive field is widely enlarged but the output still lacks structure. Nevertheless, it sounds coherently like a piano and has a feeling of virtuosity but without direction, something like the equivalent of gibberish speech.



Graphic 3.8: Dilated causal convolutions used in WaveNet effectively expanding the historical receptive field exponentially with parameters only growing linearly. Illustration taken from the WaveNet publication by van den Oord et al..

Inspired by WaveNet and its ability to produce better results with conditioning, a WaveNet-like autoencoder was introduced later on ***(Engel et al., 2017)***. The innovation in this publication lies in the introduction of a WaveNet-like encoder that offers the original WaveNet with conditioning, on the fly, without manual intervention. The authors also lay forward a huge monophonic dataset at 16 kHz with more than 300 000 four-second samples of different instruments playing held notes; three seconds actively and one second for decay. Apparently, such benchmark dataset does not exist and the authors intend for their dataset, called NSynth, to fill this hole. Each sample is also tagged by a few characteristics and the publication is of course devoted to trying their improved model on this dataset in terms of note reconstruction and instrument and pitch interpolation. These samples are longer than could be generated with good results without conditioning in the original WaveNet and since no external conditioning has been supplied, the authors argue that they construct their conditioning automatically which is a leap forward. Interpolations are done by way of interpolating between points in the latent content used to condition the (decoder) WaveNet. This is remarkable since this latent space does not seem to be structured, as it would be with a VAE for example. The authors show that their model is better than a baseline CNN (modelling FFT spectrograms) and give samples of how they can interpolate between instruments to create new timbres that sound original, even though, formally when analyzed, they show similarities with their ancestors. Optionally, the WaveNet decoder can be conditioned on pitch as well which does not improve performance, however, structure is upheld during the generated four-second output. Online fiddling with instrument timbres can be done with this model at `http://g.co/soundmaker`.

- **Generative Adversarial Network (GAN):**

  In C-RNN-GAN, the Generator is a unidirectional LSTM whereas the Discriminator is a bidirectional LSTM ***(Mogren (C-RNN-GAN), 2016)***. Each note is represented by four real values: frequency (pitch height), length (duration), intensity (velocity or dynamics) and offset to previous note. This implies that notes are modelled sequentially but polyphony can be modelled by setting offset to 0 for several successive data points. All values are normalized to be in the range $[0, 1]$ and a baseline LSTM model is used for comparing results. This is one of the first publications to use this sequential way of modelling polyphony. Samples sound vaguely like music but very unstructured and random. Measures of evaluation are devised by the author and during training, short segments are learned first, which the author refers to as a form of curriculum learning strategy. Results are evaluated according to custom measures that take different musical qualities into account.

  GANs usually work with real values for differentiation purposes and have to work with full patterns for the Discriminator versus Generator game to work. Thus, they have not been applicable to sequential problems, nor for problems involving discrete outputs, such as letters or notes. In SeqGAN, this problem is alleviated by using reinforcement learning and a policy gradient instead of the loss gradient of the Discriminator. The prefix sequence acts as the state of the Generator agent and the next token to pick is the action. To solve the partial sequence problem, the remaining sequence is averaged over multiple Monte Carlo samplings before being passed to the Discriminator. Different architectures are used for the agents and the Generator is an LSTM network whereas the Discriminator is a CNN. Residual connections are used in the latter, which also uses multiple filter sizes and pooling before deciding its verdict. The Generator and Discriminator are trained alternately and several tasks in different fields are approached. In the music field, the authors show that the SeqGAN performs better than the pretrained LSTM Generator on melodies from the Nottingham dataset, modelled as sequences of discrete numbers corresponding to a piano key ordering.

  Another take with GANs is presented in FusionGAN where the final objective is to fusion two genres in a GAN able to generate melodies similar to both ***(Chen et al. (FusionGAN), 2017)***. The idea is that two separate GANs are first trained on each of the genres by itself. A third GAN is then introducted and training is started anew whereby the new Generator is trained by both the previous Discriminators as well as its own Discriminator. It attempts to produce outputs that lie in between the previous Discriminators and at the same time, the new Discriminator is trained using both the previous Generators as well as real data and its own Generator. In this process, the initial GANs are trained as well to further improve. Discriminators are realized using TextCNNs, or 1D convolutions, and the Generators are

LSTMs. Several baselines are use for comparison and the results of FusionGAN are not strictly convincing and some of the baseline methods are similar in terms of results. Only a single unconvincing sample is available and it remains hard to determine exactly how exciting the outcome of this experiment is, even though the idea is very intriguing.

In MuseGAN, four bars are modelled at a time in a feedforward fashion using CNNs without any attention paid to recurrent mechanisms ***(Dong et al. (MuseGAN), 2017)***. The publication is thorough and focuses on multi-track polyphonic music, the former being quite rare and implying the use of many instruments, in this case bass, guitar, drums, piano and strings. In practice, it amounts to using several input piano matrices, one per instrument, and deciding how these should interact; the authors present the jamming model, where each instrument has its own Generator, Discriminator and input noise, the composer, where a single Generator generates all instruments from the same input noise (and subsequently share Discriminator) and a hybrid model that uses several Generators that have both shared and individual input noise and a common Discriminator. The different models have their instruments generated with different inter- and intra-connections and so the authors expect them to yield different results. Generated output is evaluated with custom metrics as well as with a panel whose opinions show a realistic but modest acceptance of the music, which correlates with both the analysis of the authors and what the samples sound like; there is structure and instruments are playing in idiomatic ways, but 4-bar patterns are too short to really reveal anything.

Building on SeqGAN, the originators of Objective-Reinforced GAN (ORGAN) trains the Generator by means of both the Discriminator and a reward function that takes into account the ratio of stepwise and fifthwise movements in the generated music ***(Guimaraes et al. (ORGAN), 2017)***. One might discuss whether this is a sign of qualitative music but the authors show that the resulting music contains a higher degree of these traits when tuned to do so. The Discriminator and the mentioned reward function are taken into weighted account by the value of a tuneable parameter that sums to 1; if it is set to 1, then ORGAN degrades into a SeqGAN whereas if it is set to 0, only the reward function is taken into account, resulting in a naive reinforcement learning model as opposed to a GAN. The authors keep this value at 0.5 mostly, taking both factors into account to an equal amount. As in SeqGAN, the Generator is an LSTM network and the Discriminator is a CNN network and the model is trained on MIDI melodies of 2.25 bars in size.

Continuing to build on SeqGAN, the domain was later expanded to polyphonic music ***(Lee et al. (SeqGAN), 2017)***. From the article, however, it is obvious that free polyphonic music is not targeted but instead melody and chords, more commonly referred to as homophony. The authors use a word encoding encompassing both melody and chord, represented as a set of pitches, along with duration, which is variable. The authors argue that the variable length time steps is an improvement in terms of data representation but several details on this are lacking. For example, given that each (variable) time step has both a melody note and a chord, the problem of rearticulation, which is unaddressed, is still present since one can be longer than the other. Apart from the new data representation, nothing new is added. Samples sound acceptable, but not remarkable and without long-term structure as indicated by the authors.

Inspired by the Deep Convolutional GAN (DCGAN) ***(Radford et al., 2015)***, the originators of MidiNet uses deep CNNs for both the Generator and the Discriminator in a GAN network that models melody in a feedforward style, bar by bar with an optional chord conditioning ***(Yang et al. (MidiNet), 2017)***. Conditioning is available in two main forms: as a 1D vector representing the chord of the current bar or in 2D shape representing the content of one or several preceding bars. When using a 2D shape, an extra component is added called a conditioning CNN which is also a deep CNN that produces intermediate outputs suitable to the inputs of the Generator CNN for the conditioning to blend in optimally. Three models are constructed where the first only uses previous bar(s) conditioning, the second uses chord conditioning with previous bar(s) conditioning only in the last layer and finally a model which uses full both chord and previous bar(s) conditioning to generate a bar of melody. A panel indicates that the output of MidiNet is at the same level or better than contemporary models from the Google Magenta project. The panel scores music based on how pleasing it is, how real it sounds and how interesting it is. The last category is the one where MidiNet apparently excels the most. The samples sound ok but do not stick out as state of the art.

- **Other:**

  DeepBach is a dependency network, which can be thought of as a Bayesian network with cycles, that learns using Gibbs sampling ***(Hadjeres and Pachet (DeepBach), 2017)***. The authors specialize on Bach chorales with strictly four voices and focus on harmonizations and reharmonizations of these. Each voice is modelled with a separate submodel and apart from the pitches of each voice, fermatas (the equivalence of the point or end-of-sentence in language), different subbeats of the quarter note and the time signature are taken into account. Given a time step and a voice for which we want to predict the value, each voice is modelled with one LSTM network processing a window before the current time step as well as an LSTM network working in the opposite direction from the end of the same size future window. At the current time step, a feedforward network processes the data except for the note to predict. The output of this network along with the final output of all the different voice LSTMs are then concatenated and fed into a final feedforward network that outputs a prediction. The network is trained with Gibbs sampling which is time consuming. After training, the user can determine what to hold fixed and what to resample and voices can be held fixed to only reharmonize some parts and fermatas can be introduced to induce a musical ending. DeepBach is thus a tool meant for interactive use and specifically for harmonization or reharmonization. Despite this, the authors use DeepBach to generate music from scratch, a process which is slightly unclear given the previous details, but it turns out that is it not entirely easy for a panel to determine whether generated music is real Bach or DeepBach. Results sound impressive and much like Bach.

3.2.4.8 Frameworks

Most early publications lack information on the details of how models were implemented. In one exception to this, it is written that the language C was used for implementation ***(Freisleben, 1992)***. Gang also mentions that PlaNet was used for several of his networks ***(Gang and Lehmann, 1995, Goldman et al. (NetNeg), 1999)***. Matlab and its Neural Net framework has also been used ***(Franklin and Locke, 2005)***. In the middle era, standardization in general becomes more prevalent, including standardization of preprocessing libraries as well as datasets and neural network frameworks. The latter however, seems to be evolving slower than the other two and different, more or less obscure, libraries are used. An example of this is Aspirine & Migraines written in C and C++ ***(Adiloglu and Alpaslan (NeuroComposer), 2007)***. Java has also been used for implementations ***(Bickerman et al. (RBM-provisor), 2010)***.

In the recent era, frameworks for neural networks are more standardized and a very early example that lists both what programming language was used as well as what libraries is the publication on DeepAutoController ***(Sarroff and Casey (DeepAutoController), 2014)***. The trend of including these details then become more and more prevalent over the years and numerous frameworks appear. Some frameworks are high-level and acts as a front-end to a lower level framework whereas others are self-contained.

Also in the recent era, it becomes more common to use existing frameworks for input data preprocessing. Most preprocessing frameworks processes MIDI but exceptions to this exist.

Detailed lists over publications and frameworks and preprocessing frameworks can be found in Appendix A and Appendix B.

3.2.4.9 Datasets

For most early publications as well as models that make it necessary for the training data to fulfill some very particular role, custom datasets, not always clearly accounted for, are used. Later on however, some datasets surface and start to get standardized and hence get used more often.

The Nottingham database is the first dataset used in a publication that later becomes a standard dataset for music modelling tasks ***(Eck and LaPalme, 2006)***. The same publication also uses data from The Session which, even though not a standard source of music, appears frequently. When entering the recent era, the four most used datasets in the history of music modelling with neural networks are laid forward ***(Boulanger-Lewandowski et al. (RNN-RBM, RNN-NADE), 2012)***. The reason to this is many-fold: first of all, no datasets were standardized before then and so there was a need for this, especially given that music modelling started to gain more and more attention in the community. Furthermore, many datasets required a lot of preprocessing to end up in a suitable format and since Boulanger-Lewandowski et al. made their preprocessed versions available to the public, it was easy for people to adopt this standard. Finally, their article is impressive and sets forth new baselines with results for a plethora of simpler models and so, working in the same field, it was a glorious venture to set out to improve on their work which essentially made it necessary to use their datasets. Thus, the combination of circumstances and content of their article resulted in a powerful impact and a threshold to the future being passed.

The datasets presented by Boulanger-Lewandowski et al. are MuseData, Piano-midi.de, Nottingham (which had been used before) and JSB chorales and they are, as said, available as both preprocessed MIDI and piano roll versions, partitioned into training, validation and test sets, as well ***(Boulanger-Lewandowski et al. (RNN-RBM, RNN-NADE), 2012)***. A lot of publications use the datasets in this preprocessed piano roll form whereas others use the original datasets to preprocess the MIDI themselves. Walder later tries to replicate the milestone by Boulanger-Lewandowski et al. by suggesting a new data representation, achieving (close to) state-of-the-art results on the previous benchmark datasets and making the same available in a preprocessed format more suitable to his data representation ***(Walder, 2016)***. It seems, however, that the penetrating power of this attempt, despite good results, was not at the same level as that of the accomplishment by Boulanger-Lewandowski et al..

Also purely signal-based datasets are available with audio signals instead of a data representation.

Some preprocessing frameworks for MIDI or some other representation have datasets included in a ready-to-use fashion.

A detailed list over publications and used datasets can be found in Appendix C.

3.2.4.10 Evaluation

No standardized way to evaluate generative networks modelling music exists and contributors have focused to a highly varying degree on this issue; some go through great elaboration to try to devise measures of success on their own ***(Coca et al., 2011; Liu and Ramakrishnan, 2014; Mogren (C-RNN-GAN), 2016; Colombo et al. (DAC), 2017; Dong et al. (MuseGAN), 2017; Guimaraes et al. (ORGAN), 2017; Sabathé et al. (DRAW), 2017)*** whereas others simply use what others have used. In a lot of cases, measuring test set loss of known datasets (or similar measures) and comparing to other models is used ***(Franklin, 2004; Franklin and Locke, 2005; Franklin, 2006; Franklin, 2005; Boulanger-Lewandowski et al. (RNN-RBM, RNN-NADE), 2012; Goel et al. (RNN-DBN), 2014; Liu and Ramakrishnan, 2014; Gan et al. (TSBN), 2015; Lyu et al. (LSTM-RTRBM), 2015; Vohra et al. (LSTM-DBN), 2015; Madjiheurem et al. (Chord2Vec), 2016; Sturm et al. (char-rnn, folk-rnn), 2016; Walder, 2016; Yu et al. (SeqGAN), 2016; Agarwala et al., 2017; Bretan et al., 2017; Johnson (LSTM-NADE, TP-LSTM-NADE, BALSTM), 2017; Mehri et al. (SampleRNN), 2017; De Prisco et al., 2017; Engel et al., 2017; Hadjeres***

and Nielsen (Anticipation-RNN), 2017; Hennig et al. (Classifying VAE, Classifying VAE+LSTM), 2017; Huang et al. (CocoNet), 2017; Roberts et al. (MusicVAE), 2017; Roberts et al. (MusicVAE), 2018; Tikhonov and Yamshchikov (VRASH), 2017; Wu et al. (HRNN), 2017; Ycart and Benetos, 2017; Colombo and Gerstner (BachProp), 2018; Walder and Kim (MotifNet), 2018), especially so in publications treating music as a benchmark problem ***(Bengio et al., 2012, Bayer et al., 2013,Pascanu et al. (DT(S)-RNN, DOT(S)-RNN, sRNN), 2013; Chung et al., 2014; Bayer and Osendorfer (STORN), 2015)***. Formulating some baseline models oneself and then proving the capacity of one's contribution is a yet more efficient way to evaluate since it guarantees a uniform data processing and eliminates sources of errors due to this ***(Franklin and Locke, 2005; Eck and LaPalme, 2006; Boulanger-Lewandowski et al. (RNN-RBM, RNN-NADE), 2012; Chung et al., 2014; Sarroff and Casey (DeepAutoController), 2014; Sigtia et al., 2014; Nayebi and Vitelli (GRUV), 2015; Bretan et al., 2016; Chu et al., 2016; Huang and Wu, 2016; Jaques et al. (RL Tuner), 2016; Kalingeri and Grandhe, 2016; Lattner et al. (C-RBM), 2016; Madjiheurem et al. (Chord2Vec), 2016; Mogren (C-RNN-GAN), 2016; O'Brien and Román (MusicNet), 2016; Sun et al., 2016; Walder, 2016; Yu et al. (SeqGAN), 2016; Agarwala et al., 2017; Chen et al. (FusionGAN), 2017; Johnson (LSTM-NADE, TP-LSTM-NADE, BALSTM), 2017; Mehri et al. (SampleRNN), 2017; Engel et al., 2017; Guimaraes et al. (ORGAN), 2017; Hadjeres and Nielsen (Anticipation-RNN), 2017; Hennig et al. (Classifying VAE, Classifying VAE+LSTM), 2017; Huang et al. (CocoNet), 2017; Jaques et al. (Sequence Tutor), 2017; Roberts et al. (MusicVAE), 2017; Roberts et al. (MusicVAE), 2018; Shin et al., 2017; Tikhonov and Yamshchikov (VRASH), 2017; Wu et al. (HRNN), 2017)***. Another popular approach, orthogonal to the previous ones, is to use the verdict of a panel of listeners ***(Hild et al. (HARMONET), 1991; Gang and Lehmann, 1995; Hörnel and Menzel (MELONET), 1998; Eck and Schmidhuber, 2002; Melo and Wiggins, 2003; Bretan et al., 2016; Chu et al., 2016; Huang et al. (ChordRipple), 2016; Huang and Wu, 2016; Kalingeri and Grandhe, 2016; Sturm et al. (char-rnn, folk-rnn), 2016; Yu et al. (SeqGAN), 2016; Agarwala et al., 2017; Chen et al. (FusionGAN), 2017; Mehri et al. (SampleRNN), 2017; De Prisco et al., 2017; Dong et al. (MuseGAN), 2017; Hadjeres and Pachet (DeepBach), 2017; Huang et al. (CocoNet), 2017; Jaques et al. (Sequence Tutor), 2017; Liang et al. (BachBot), 2017; Malik and Ek (StyleNet), 2017; Roberts et al. (MusicVAE), 2018; Shin et al., 2017; Wu et al. (HRNN), 2017; Yang et al. (MidiNet), 2017; Colombo and Gerstner (BachProp), 2018; Mao et al. (DeepJ), 2018)*** and perhaps perform some kind of blind test whose outcome is a measure of success. During the recent era, a test informally referred to as a musical Turing test is sometimes used, whereby participators are given two samples of music and are asked to point out the one written by a human being ***(Malik and Ek (StyleNet), 2017; Shin et al., 2017; Wu et al. (HRNN), 2017)***. Applying statistical methods or frameworks to validate whether the output, according to some measure, is not random is also common ***(Melo and Wiggins, 2003; Sigtia et al., 2014; Bretan et al., 2016; Huang et al. (ChordRipple), 2016; Huang and Wu, 2016; Jaques et al. (RL Tuner), 2016; Lattner et al. (C-RBM), 2016; Mogren (C-RNN-GAN), 2016; Sun et al., 2016; Yu et al. (SeqGAN), 2016; Bretan et al., 2017; Chen et al. (FusionGAN), 2017; Hadjeres et al. (GLSR-VAE), 2017; Hadjeres and Nielsen (Anticipation-RNN), 2017; Hennig et al. (Classifying VAE, Classifying VAE+LSTM), 2017; Jaques et al. (Sequence Tutor), 2017; Lee et al. (SeqGAN), 2017; Liang et al. (BachBot), 2017; Roberts et al. (MusicVAE), 2018; Sabathé et al. (DRAW), 2017; Shin et al., 2017; Ycart and Benetos, 2017; Eppe et al., 2018)***. In some straightforward tasks, measuring the accuracy of the trained model against a somewhat exhaustive test set is enough ***(Laden and Keefe, 1989; Mozer (CONCERT), 1990; Mozer and Soukup (CONCERT), 1991; Mozer (CONCERT), 1994; Eck and Schmidhuber, 2002)***.

In some cases, the expectations on the model is very clear and conforms to some well-defined standard and in those cases, it is easier to evaluate in a meaningful way. An example of this is systems that should be able to alternate with a jazz soloist in playing solos ***(Nishijima and Watanabe (Neuro-Musician), 1993; Franklin (CHIME), 2001; De Prisco et al., 2017)*** which have been evaluated by letting them perform this very task with live players approving, at least in part, of the result. Other examples are NetNeg outputting first species counterpoint which is a very strict and clear way of writing music ***(Goldman et al. (NetNeg), 1999)***, or models outputting jazz chord progressions ***(Choi et al. (char-RNN, word-RNN), 2016)*** which can easily be determined to be in style or not. In WaveNet where speech is formed, it is easy to determine if it makes sense or not and the same goes, to a limited extent, for the piano music produced by the same ***(van den Oord et al. (WaveNet), 2016a)***. Having real musicians play music from a model also, to some extent, indicate that the output is of quality ***(Sturm et al. (folk-rnn, char-rnn), 2016; Hadjeres and Pachet (DeepBach), 2017; Colombo and Gerstner (BachProp), 2018)***. For models learning latent representations of the material, the reconstruction loss is a very simple way to evaluate the success ***(Large et al., 1995)*** and in transcription contexts, it is very easy to determine whether a music model performs correctly and improves transcriptions or not ***(Sigtia et al., 2014)***.

In Appendix D is presented an overview of the most comparable results achieved by different publications, namely those achieved on the standard datasets, as set forth by Boulanger-Lewandowski et al. ***(Boulanger-Lewandowski et al. (RNN-RBM, RNN-NADE), 2012)***. This publication uses a piano roll data representation that they have made available to the public for easy comparative results. They also specify their loss as the average loss of a single time step. Over time, however, other publications aspiring to set new results on these datasets do not always use these preprocessed versions and this results, of course, in uncertainties regarding the comparability between different results.

3.2.4.11 Source Code

Making the course code available to the public, in the spirit of truly scientific methods, is a trend that has increased monotonously over the years. One reason for this is the standardization of frameworks and tools which make it easier for anyone to replicate an experiment. Nevertheless, even as of today, far from all source code is available for the general public.

One of the first to release source code that is still available today is Bickerman et al. ***(Bickerman et al. (RBM-provisor), 2010)***.

A detailed list of publications and links to their source code can be found in Appendix E.

3.2.4.12 Samples

One might find it self-evident that publications about systems that should be able to compose music in some way provides samples of this as part of a results section. Unfortunately, and to much disappointment, this is not always the case, even though this tendency has improved over there years.

Many early publications show sheet music as part of the publications ***(Todd, 1989; Mozer (CONCERT), 1990; Hild et al. (HARMONET), 1991; Lewis (CBR), 1991; Mozer and Soukup (CONCERT), 1991; Freisleben, 1992; Tsang and Bellgard (EBM), 1992; Bellgard and Tsang (EBM), 1994; Mozer (CONCERT), 1994; Gang and Lehmann, 1995; Gang and Berger, 1996; Large et al., 1995; Goldman et al. (NetNeg), 1999; Berger and Gang, 1997; Gang et al. (HNN), 1997; Gang et al. (HNN), 1999; Hörnel and Menzel (MELONET), 1998; Chen and Miikkulainen, 2001; Franklin (CHIME), 2001)***. This holds true for the middle period as well ***(Eck and Schmidhuber, 2002; Verbeurgt et al., 2004; Franklin, 2006; Franklin, 2005; Adiloglu and Alpaslan (NeuroComposer), 2007; Corrêa et al., 2008; Bickerman et al. (RBM-provisor), 2010; Coca et al., 2011)*** even though it now gradually becomes more common to supply online links to listening samples as well. In the recent period, written music is often presented in the publications ***(Liu and Ramakrishnan, 2014; Sun (DeepHear), 2015; Bretan et al., 2016; Choi et al. (char-RNN, word-RNN), 2016; Chu et al., 2016; Colombo et al., 2016; Huang et al. (ChordRipple), 2016; Sturm et al. (char-rnn, folk-rnn), 2016; Walder, 2016; Colombo et al. (DAC), 2017; Johnson (LSTM-NADE, TP-LSTM-NADE, BALSTM), 2017; Dong et al. (MuseGAN), 2017; Hadjeres et al. (GLSR-VAE), 2017; Hadjeres and Pachet (DeepBach), 2017; Hadjeres and Nielsen (Anticipation-RNN), 2017; Teng et al., 2017; Wu et al. (HRNN), 2017; Yang et al. (MidiNet), 2017; Koh et al. (C-RVAE), 2018; Mao et al. (DeepJ), 2018)*** even though it is desirable that this is complemented with links to online listening samples. At this time, it also gets more and more common to publish reconstruction samples in piano roll representation that are comparable to used training samples ***(Sigtia et al., 2014; Gan et al. (TSBN), 2015; Lyu et al. (LSTM-RTRBM), 2015;***

Lattner et al. (C-RBM), 2016; Malik and Ek (StyleNet), 2017; Roberts et al. (MusicVAE), 2017; Roberts et al. (MusicVAE), 2018; Ycart and Benetos, 2017; Simon et al. (MusicVAE), 2018).

The analogy of showing samples of written music in the context of modelling the actual audio signal is showing spectograms ***(Sarroff and Casey (DeepAutoController), 2014; Nayebi and Vitelli (GRUV), 2015; Engel et al., 2017; Eppe et al., 2018)***.

Some early exemplary publications with Internet links are unfortunately expired and the links are either dead ***(Hörnel and Menzel (MELONET), 1998; Eck and LaPalme, 2006)*** or miss the referred content ***(Chen and Miikkulainen, 2001)***.

Publications in the recent era should, to be convincing, publish both written samples as well as listening samples online, as a complement to more formal evaluation. In these modern days, there is really no excuse for not making such data available, especially since there are examples of publications that show good formal evaluation accompishments but whose samples sound really bad, thus, there is not a one-to-one mapping between good scores on evaluation metrics and the ability to generate somewhat convincing music, thus urging the need for both. Examples of publications that aim at showing great state-of-the-art results, or simply insinuate that their models perform well, but lose in credibility due to the lack of samples are ***(Goel et al. (RNN-DBN), 2014; Vohra et al. (LSTM-DBN), 2015; Colombo et al., 2016; Huang and Wu, 2016; O'Brien and Román (MusicNet), 2016; Walder, 2016; Hadjeres et al. (GLSR-VAE), 2017; Hadjeres and Nielsen (Anticipation-RNN), 2017; Walder and Kim (MotifNet), 2018)***. Some publish samples that are decorated by using very elaborate synth sounds with effects and adding drum tracks in which case it is easy to get mislead from the actual output of the model and instead get impressed by the added surface.

A detailed list of publications and links to samples are available in Appendix F.

3.3 Summary

Depending on how we restrict the notion of algorithmic composition, it has been around for a very long time and ever since the dawn of computers, a certain interest has also existed around how to involve these machines in the task. In general, the history of algorithmic composition is diverse and many different directions have been, and are still, taken, both mathematical, random and grammatical only to mention a few. Some of the most successful results seem to have been acquired with hybrid systems.

Neural networks have been used to model music since the 80's and at least two paradigm shifts and a certain evolution covering most, but not all, aspects can be followed. To a high degree, the evolution follows that of general neural networks with deep learning on the rise in the last decade. Another, less expected, tendency is that the level of musical knowledge among authors seems to have dropped over the years. Maybe this is expected in one way since it indicates that algorithmic composition with neural networks have been sucked up into the corpus of common neural network fields which diverse authors try their models in without necessarily being musicians or knowledgeable in music. Different schools of thought emerge in several ways and one example is the strong preference for simple layouts that should work with minimal guidance counterbalanced with complex frameworks with lots of subcomponents that requires human interaction. Another tendency is that some authors regularly publish new articles with updates and improvements of previous models whereas others try out new directions every time.

Given that this is a literature survey on algorithmic composition with neural networks, it is not unexpected that most of the studied publications are on networks that compose. Some networks do related tasks such as harmonization and the farthest away from this core task we get is when networks are used as a prior for automatic music transcription (AMT). It is evident that music models can also serve a purpose in the field of music information retrieval (MIR) even though it is out of the scope of this thesis. The structure of the music modelled has always been diverse and continues to be so. A development is that it is more common to model polyphonic music more recently but even so, monophonic and homophonic music are still modelled today.

As far as data processing is concerned, authors started off making hand-crafted features highly differing between publications but it has over the years become more and more common to use MIDI as an input representation. The reasons for this are many-fold and one reason is that the direction taken towards deep learning and deeper models makes the availability of large volumes of training data necessary. MIDI can be recorded from performances as well as be generated from synthesizers and software and even though the mass of MIDI files on the Internet is diverse, both in terms of genre, quality and specific encoding, it is probably the largest free source of (somewhat) symbolic music possible to convert to a reasonable data representation available. The MIDI format is also very rich, putting few restrictions and assumptions on musical properties and thus suitable for different kinds of modelling. This richness has the downside that it allows for properties, such as expressive timing, which might make conversion to an internal data representation somewhat cumbersome and ambiguous. Some publications pay extra attention to this and there are software libraries that automate the task. Nonetheless, this conversion may be a source of uncertainty when it comes to comparing results and up to this date, no standard exists.

When it comes to internal representations, an intuitively appealing, and throughout time the most common, way to represent music is by means of a matrix with time along one dimension and pitch along another; along the time axis, time slices corresponding to a chosen fraction of time or note values float whereas in the pitch axis, an ordered chromatic scale unfolds. There are at least two problems with this representation. The first problem is that one cannot distinguish a held note from a repeated one and the second one is that outputting a polyphonic prediction for an entire time slice at a time unavoidably results in the output notes being conditionally independent, which is not very realistic. The first problem can be solved by adding extra inputs that account for rearticulation and the second problem is handled by allowing the output function to be handled by a mechanism that do not make assumptions on conditional independence, for example a NADE or an RBM. Another problem with time slicing, however, is also that the finer the granularity, the more the model is rewarded for repeating the same pitch over and over again, which probably has negative impact on the generative capabilities, albeit not on the automatic evaluation metrics. Even though less intuitive and easy to overview, other representations have had more success, for example modelling one note at a time in a sequential manner, which works both for polyphonic and monophonic music, and makes the MIDI representation even more similar to the internal representation used.

The sequential aspect of music has mostly, adequately, resulted in the employment of RNNs to model music, no matter if a number of time steps are used to predict the next or if each time step is predicted in a continuous manner, with a time step constituting everything from a single event to a bar of music. Even later on, now instead using GRUs or LSTMs, different sorts of RNNs have continued to be the most common basis for networks modelling music. In conjunction with this, RNNs might stick out as opposites to feedforward networks but both have actually been used and it is not hard to see why; RNNs have an unlimited temporal scope which is in reality limited and feedforward networks may be equipped with extra inputs accounting for a number of contextual units making them somewhat temporal. One direction of development has been to use other architectures for output (for example an RBM or NADE) whereas another one has been to use RNNs in sequence to sequence models instead, inspired from language modelling. It is noticeable how undirected models (RBMs and DBNs) continue to provide solid results despite their age and less transparent underlying theory. Embedding layers also improve most results as a rule. Early on, the networks were often guided in terms of additional inputs as well as through different architectural choices in the model. The originators of articles were then, as mentioned, often musicians, or musically interested, themselves and thus had better possibilities to thoroughly analyze how to present the different aspects of music to a network. The tendency has ever since been the same as in other areas of neural network learning: larger networks, new network architectures, more training data and less, or even no, guidance. Typically, it is desirable in many paradigms that a model should be able to be trained end-to-end with only the data. Nevertheless, there are still authors who believe that music modelling with neural networks should be done in tandem with a user guiding the process somehow. Depending on what expectations that is reasonable to have on these algorithms, both standpoints can be defended. Apart from previously mentioned models, entirely different strategies have also been used with feedforward and convolutional networks. Given a more abstract view on the layout of architectures, the notion of different hierarchies has become more and more common recently with the goal that a model should be able to discover and account for structures at different hierarchical levels. With hierarchical models comes also the importance of determining the direction of conditioning, given that some level of hierarchies can be made dependent on others. Along a third line of thought, networks that are partitioned into subnetworks might be used where some subnetworks might not even be neural networks. Thus, the general evolution of architectures advocate to add potential to the task either using more expressive and advanced state-of-the-art neural network devices or to connect traditional architectures together, either as a single large network or multiple smaller.

The level of transparency has also increased over the years and it is nowadays common, in the true spirit of science, to make available both code and listening samples of the accomplished architectures. However, an area where no unified standard has emerged is in that of evaluation. Here, we see a trend where general data scientists tend to use validation set losses on benchmark datasets to measure the level of success whereas data scientists with a more musical background tend to focus more on panels of listeners from different backgrounds that get to express their thoughts on generated music. Both obvious and less obvious problems exist with both of these methods and in the case of benchmark datasets, it is hard to know whether comparisons of results between publications are fair or not. Authors referring to results on benchmark datasets also often pay little to no attention to what the generated music actually sounds like and in the few cases where samples are actually available (because in this category, they tend not to be), the results do not correlate with musical quality. Different evaluation hybrids exist as well where customs statistical metrics are used to measure the quality of the output, often with the result that the chosen musical parameters seem arbitrary and not necessarily indicative of quality. One remarkable observation is that overfitting is a tendency discussed to virtually no extent when compared to other fields where neural networks are used. It has lately been suggested that machine learning algorithms in general should be evaluated and improved with respect to performance in the original problem domain and with input from practitioners in that domain ***(Sturm et al., 2018)***. The authors put together a concert program with music, sometimes manually completed after generation, by trained models played by live musicians with an ensuing evaluation. In this context, it is proposed that machine learning in some areas become a problem in a closed world with training sets and where the end goal is to produce results in the machine learning community, missing the connection back to the original problem domain. The authors further discuss the role of machine learning in music and whether it is reasonable to aim for a standalone product or more of an aiding tool for composers and contrast large monolithic programs with several smaller working together iteratively. Everything else aside, one must not forget that music is an art and some of the problems of evaluation and keeping a steady course for the future lie in this fact.

The most prevalent unsolved problems when it comes to music modelling with neural networks are the problems of evaluation and long-term structure. A lot of suggested solutions have been offered for both but in the case of evaluation, none has become standardized. As far as long-term structure is concerned, different architectures and processing methods have been used, sometimes with promising results but not to the extent that the problem has really been solved. Given the fact that many authors state that repetition is also a significant property of music ***(Freisleben, 1992; Eck and LaPalme, 2006)***, it is not hard to understand that long-term structured repetition is crucial for a neural network to model music with satisfactory results.

4 METHOD

In this section, design choices are first made and motivated after which is presented the architecture that was implemented and used for the remaining parts of this work.

4.1 Design Choices

First of all, a few thoughts on some very common tendencies encountered in other work. For example, it is often implicitly conveyed that the more intuitively appealing to the human brain a modelling pattern is, the better, even though this might not at all be the case. Even though inspired by the human nervous system, and therefore the brain, neural networks don't function in the same way as the human brain and when presented with indications that a non-intuitive representation is better, these must be taken seriously. Similarly, often some aspect of the human compositional process is attempted to be ported to a machine learning model whereas in reality, there is really no indication of this being a winning strategy; maybe we have to take into account that neural networks compose in a different way than we do? Finally, the similarities with text and music is often emphasized which can be questioned. After all, in music a single motive can be the basis for an entire piece and so music shows much more self-similarity than does a general text, possibly with the exception of some particular types of text, for example poetry or lyrics.

Based on the work accounted for in section 3, several design choices are to be made with respect to architecture. As a starting point for this, ten models from the literature survey are picked out as the state of the art. These models are then further refined in number and then used to guide a series of choices.

4.1.1 State of the Art

The following models from section 3 are considered the state of the art because they stand out thanks to interesting design choices and / or well-sounding results:

- **C-RBM *(Lattner et al., 2016)*:**

  Being one of the relatively few examples where a convolutional network is used to model music, the C-RBM builds on the attractive idea of iterative improvement and is the only one of its kind. It builds on rather complex theoretical grounds using an old-school RBM but its generated music sounds interesting.

- **PerformanceRNN** ***(Simon and Oore, 2017)*****:**

  Now being a part of the Google Magenta project, the PerformanceRNN takes a step away from the discretized data representation and provides impressive sounding results despite the more unintuitive, yet MIDI-like, representation with on and off events for single pitches and separate inputs that move time ahead.

- **BALSTM** ***(Johnson, 2017; De Prisco et al., 2017; Mao et al. (DeepJ), 2018)*****:**

  Giving a good solution for how to preserve the musical property of transpositional invariance and refrain from augmenting the training data with transpositions in all keys, the BALSTM produces music that sounds nice with convincing architectural arguments. The original author effectively shows that the transpositional invariance property is present ***(Johnson, 2017)*** and the architecture also lends itself to use along with additional properties and conditioning ***(Mao et al. (DeepJ), 2018)***.

- **DeepBach** ***(Hadjeres and Pachet, 2017)*****:**

  With a complex machinery of several diverse subcomponents, mixing unsupervised and supervised learning, DeepBach provides outputs in a limited musical context and genre that sounds very impressive.

- **BachBot** ***(Liang et al., 2017)*****:**

  Employing the somewhat same limited musical context as DeepBach, BachBot uses time slicing but with the innovative novelty of modelling the polyphony in a time-sliced step sequentially. The architecture is much simpler than DeepBach and uses less spread ideas in terms of machine learning but its results sound equally well.

- **CocoNet** ***(Huang et al., 2017)*****:**

  Also excelling in the narrow field of Bach chorales, CocoNet provides great sounding polyphonic results from a rather simple convolutional architecture built on rather more complicated theoretical grounds.

- **JamBot** ***(Brunner et al., 2017)*****:**

  With a simple architecture used in an innovative way, JamBot produces virtually unrestrained polyphonic music that sounds good while using one LSTM for chord conditioning in an interesting way.

- **MusicVAE** ***(Roberts et al., 2017; Roberts et al., 2018; Simon et al., 2018)*****:**

  Noticeable because it forms an important part of the Google Magenta project, the MusicVAE also provides results that are both intriguing and inspiring to listen to. Being one of the state-of-the-art architectures, it has been used in several papers and provides good results both when being used with a time slicing data representation ***(Roberts et al., 2017; Roberts et al., 2018)*** as well as a non-discretized one ***(Simon et al., 2018)***. The musical property of structural hierarchies are also manifested in the very design of this model.

- **BachProp** ***(Colombo and Gerstner, 2018)*****:**

  Bringing attention to the importance of training data and data representation, the originators of BachProp demonstrates a simple and efficient preprocessing pipeline for MIDI while also modelling several parameters in a non-discretized and novel manner with internal conditioning among the properties in every time step in an impressive way. The demonstration on data representation also accentuates the question of what impact improper data preprocessing has had historically on models trained on MIDI.

- **MotifNet** ***(Walder and Kim, 2018)*****:**

  Another complex architecture with many subcomponents but with very convincing theoretical ideas that build on musical properties. It sticks out from the mass but lacks both technical detail and listening samples.

4.1.2 Paragons

The state of the art is now further reduced in number to arrive at a few paragons, or role models, whose design choices are worthy to look into and carry over to the new architecture that is to be implemented.

As simplicity is a general virtue and a guide line in the hunt for paragons, DeepBach, C-RBM and MotifNet are early discarded as such. The MotifNet paper also lacks technical detail to the degree where drawing anything but highly abstract inspiration from it would be infeasible. Furthermore, the success of convolutional networks in algorithmic composition with neural networks has been somewhat limited and it hasn't been a track that a large portion of authors have chosen to head down. For this reason, C-RBM and CocoNet are discarded as paragons since using them would imply too much of restriction in other design choices, e.g. experiences from the vast majority of RNN models would go lost. Models resting on older theoretical grounds are also less suitable as modern paragons which are also reasons for discarding C-RBM and DeepBach. JamBot is furthermore discarded because it builds on the notion of chords as an integral component, which is a liability since all kinds of music do not have specific chords that can represent the pitch content of a bar. It also violates condition C.6 of the problem statement since it makes assumptions on the input music and finally, it would require extra annotations (or preprocessing steps) wherever chord information is not readily available. This results in the five paragons PerformanceRNN, BALSTM, BachBot, MusicVAE and BachProp.

4.1.3 Large-scale Design - Architecture

All of the conditions in subsection 1.2 can be met with at least one of the five paragons and they shall now be used as inspiration for the network architecture to implement. Some of the conditions from 1.2 are now reiterated with the paragons in mind to get closer to an architectural choice. The remaining conditions are more closely related to *how* to build and use the model and oppose to *what* will constitute the model and we shall see that they are respected as well.

C.3 *Steerable by means of seeding or conditioning to some degree, in the way that it outputs different compositions as a result of different seeds and / or conditioning.*

Even though conditioning can be applied to any model, such as is done in DeepJ with additional inputs representing some conditioning class distribution, it hardly beats the opportunities for steering and interpolation that the structured latent space of a VAE offers, as it is a concept built into its very core. Among the paragons, only the MusicVAE contains a VAE and even though this design, in its original form, does not meet all the conditions in 1.2, it is a good base architecture, both thanks to this subcomponent but also due to its results. The MusicVAE is also conceptually simple and builds on newer machine learning advances, which are both desirable properties. The original MusicVAE summarizes an input unit (for example one or two bars) of music in time-sliced form with a bidirectional LSTM which is then used to parameterize a distribution from which a latent vector is sampled. A decoder is then used to reconstruct the input from this vector and when trained, samples and interpolations can be drawn from the latent space to generate new music similar to what the MusicVAE was trained on. In essence, it is a sequence to sequence (seq2seq) model. Additional conditioning can be introduced to any VAE by adding information to both the parameterization of the latent space distribution as well as to the decoder which ideally causes the VAE to form different latent spaces for different conditionings, which a decoder then uses along with the initial conditioning to restore the input ***(Sohn et al., 2015)***. It has been shown in the realm of algorithmic composition with neural networks that the VAE effectively uses the conditioning as conditioning only without redundant modelling of the conditioning content itself into the latent space ***(Teng et al., 2017; Simon et al. (MusicVAE), 2018)***. Arbitrary extra conditioning, additional to the attractive properties of a structured latent space, is thus easily leveraged with the MusicVAE.

C.4 *Have no restrictions on the length of the output music.*

The MusicVAE in its original form encodes music of fixed lengths (2-, 8- and 16-bar fragments) whereas the model to be implemented is supposed to be able to model music of arbitrary length. One general idea to remedy this with a VAE is to model sequences with one VAE at every time step ***(Chung et al. (VRNN), 2015)*** instead of in a seq2seq fashion. However, this would be a step too far away from the MusicVAE architecture which has proven to be good for music. The use of a VAE at every time step in a recurrent model would effectively imply sampling the next time step based on either only the state of the RNN or additionally based also on the previous step. If we consider chunks of music larger than a single time step, we can mimic this in the MusicVAE by using conditioning on some size musical context which is also passed as conditioning to the decoder. Even though this context would have some fixed (perhaps relatively small) receptive field of the musical history (context) it would provide at least some context. To the contrary, RNNs have an unlimited receptive field in theory, but it is commonly known that the temporal scope of RNNs in practice is far from unlimited.

As an example, by supplying a unit A as conditioning context while reconstructing unit A + 1, we can train a MusicVAE-like model so that after training, B + 1 is sampled and decoded with a given context B (seed or some start token), after which we in the next iteration use B + 1 as context and reiterate the sampling and decoding to reconstruct B + 2 and so on. With this, we can, at least in theory, sample music of arbitrary length.

C.1 *Model polyphonic music.*

C.2 *Model a variable number of different instruments.*

Given the basic design borrowed from MusicVAE, conditions C.1 and C.2 are not integral to the overall design but instead to the layout of the decoder. The original MusicVAE models monophonic music in a time slicing fashion and applies a predetermined number of decoding units when modelling polyphony ***(Roberts et al., 2017, Roberts et al., 2018)***. Later uses of the MusicVAE take a step away from the time slicing and model polyphony in each decoder unit but still makes use of a predetermined number of decoders for different instruments ***(Simon et al., 2018)***. To fulfill the two above conditions, there is an imminent need to pick a new decoder model rather than use the original MusicVAE decoder which places a predetermined assumption on the number of simultaneous voices or instruments in use, which violates condition C.2. Out of the paragons, both BachProp, BALSTM and BachBot are candidates but since the BALSTM provides transpositional invariance, it is superior to the others, which are therefore discarded as decoders. This design choice is extra important when modelling music that modulate since transposing all music to the same key, which is often done as a countermeasure in models that are not transposition invariant, then becomes an arbitrary operation; ideally we would like to transpose all constituent parts to the same key but this would require extra preprocessing and annotations. In some publications, all music is transposed to all or several keys to remedy the problem of transposition but this is intuitively inefficient since there is really no new information coming from the transposed music and thus, the model has to learn seemingly different patterns that are innately the same. With transposition invariance then comes less redundant training data and the ability for a model to modulate and still fully use what it has learned, but with reference to different keys. At the moment, only the BALSTM has this ability and since the final goal is to model Mahler symphonies, which are highly modulatory, the property of transposition invariance is highly valuable and so the BALSTM will be used as a decoder in the model to come, along with a baseline RNN decoder for comparison. The BALSTM has also been modelled with extra output units to account for rearticulation ***(Johnson, 2017)*** which is suitable since we might, depending on the internal data representation, need extra outputs for additional properties as well. By adding extra output units to model instrument, there is no theoretical limit to which instrument is chosen for each note and each note will have an instrument without placing an assumption on the total number of instruments to model at every time step. Both the BALSTM and a baseline RNN decoder can model polyphony, either sequentially or simultaneously with varying results and both can also have the choice of instrument added as an extra output and so both fulfill conditions C.1 and C.2.

4.1.4 In-depth Design - Implementation Details

The aspects discussed for all models in section 3 are now reiterated and treated with respect to the upcoming design choices.

- **Application / Purpose**: Composition

  As with most models, the main application of the new architecture is composition.

- **Domain**: Symbolic

  The targeted domain is symbolic music in its basic form as per condition C.7 in subsection 1.2. This means that music is represented as a series of events with pitch and duration placed along a time line and played by an instrument (timbre). Dynamics and micro-timing and other aspects are thus not attempted to be modelled and furthermore, only instruments with pitch are modelled, excluding drums and other percussion that lacks pitch as per condition C.8.

- **Musical domain**: Polyphony

  The architecture in design should model free polyphonic music with an unlimited and flexible number of simultaneous voices as per condition C.1 in subsection 1.2

- **Genre**:

  The aim is to model all kinds of Western pitch-based music with the restriction to Western music being inherent in the choice of available pitches and intervals as well as which instruments are available. Since the final goal is to model late romantic symphonies, some design choices have to be made placing some, but yet minimal, assumptions on the music, as per condition C.6.

- **Input representation**: MIDI

  This format has, by far, been the most common input representation in the history of algorithmic composition with neural networks, and the few baselines that exist are with reference to MIDI datasets. Given that symbolic music is to be modelled, a symbolic input representation, such as MusicXML or ABC, might ideally have been more appropriate. However, due to the availability of music in MIDI format, it will be used as input representation. Mahler symphonies are available online in MIDI and using datasets that have previously been used is also a great asset. The MIDI format is also well-used, in general, which makes different MIDI synthesizers and utilities more common than corresponding for other formats. To respect condition C.5, MIDI music must also be processed, as is, by a preprocessor, not requiring any other manual intervention or extra annotations.

- **Data representation**:

  The data representation is often seen as the most important part ***(Mozer, 1990; Mozer and Soukup, 1991; Mozer, 1994)*** and it is sometimes stated that together with the choice of dataset, it contributes more to success than all other aspects of an architecture ***(Simon and Oore (PerformanceRNN), 2017)***.

  - **Pitch:**

    Pitch should be represented in an absolute, local manner as has been the most common way to represent it. Binary representations also generally work well with neural networks. All the 128 pitches of the MIDI format need not be modelled but given that a range of instruments should be modelled, the range of modelled pitches can not be shrunk too much. Since the BALSTM is intended to be used as decoder, no transpositions of training data will be done.

  - **Chords:**

    Supporting free polyphony, chords will not be modelled in any particular manner but only as a collection of pitches sounding at the same time. Thus, all chords, or combinations of pitches, are possible. This is a step towards placing minimal assumptions on the music, as per condition C.6.

  - **Percussion:**

    Percussion with no pitch will not be modelled. Hence, living up to condition C.8.

  - **Duration:**

    Out of the previously given paragons, both PerformanceRNN and BachProp use a representation of time that does not build on time slicing. The MusicVAE has also been used along with a similar representation as is used in PerformanceRNN ***(Simon et al., 2018)***. Roughly estimated, the absolutely most common way to represent duration is by means of time slicing, and given that two (or three depending on how you see it) out of the five paragons use a different representation indicates that, even though highly intuitive for a human being, time slicing might not be optimal for a neural network. One might have the impression that it is "harder" for a network to learn from a sequence of events with no inherent pattern in offset and duration, and that time slicing solves this elegantly, but when presented with such coincidences as described above, attention must be paid. The downsides with time slicing, namely that we need to distinguish between rearticulations and sustained notes, along with the fact that time slicing implies that the network has to learn how to repeat the same note for long sequences, which is a non-musical property introduced synthetically into the training process, might be the reason for this. Taking inspiration from the PerformanceRNN and especially BachProp, a local absolute representation of duration and offset will be used. In both these models, simultaneous pitches are represented sequentially with zero offsets in between (sequential polyphony), which is a feature that will also be adopted. Since we model music in the symbolic domain and more specifically, written music in its basic form, durations should be given in terms of note values, and not in absolute time (e.g. in ms.).

- **Conditioning / additional inputs:**

  One might argue that the more accessible, via conditionings and additional inputs, the training data is made, the more likely it is that an algorithm will take notice of exactly the patterns that are made obvious to it. This might be both desirable and not depending on what we are trying to do. For example, if the goal is to compose music similar to the training data revealing as much as possible about its structure might be a good idea, but if we instead aim for the network to discover very abstract and latent features that we might not even be aware of ourselves, training a network with the data only, as is, might be a better idea. One way to hypothesize is that with the latter setup, we aim to a higher extent to generate progressive, new music, superficially dissimilar to the input. In this work, the ambition is positioned somewhere in between these extremes and it is attempted to generate music somewhat in an existing style, or at least in a mix of existing styles. Subsequently, some basic conditioning rooted in elementary properties of traditional music will be supplied to the network.

  Relationship to underlying metric structure, beats, should be given as conditioning as is done in many cases, to help correlate emphasized beats with the more frequently used pitches from the scale. This might be even more important when not using time slicing since the underlying meter, to some extent, is implicitly expressed with each beat always containing the same number of slices.

  Inference should be done by providing a context unit and then sampling from the VAE and decoding. Alternatively one could also initially generate a latent vector from a certain input unit instead of sampling it. After the seeding, the model can continue to run by providing the reconstruction as context and then sample from the VAE and decode anew.

• **Model / architecture**:

This section is omitted for now and deferred to subsection 4.2.

• **Frameworks**:

Implementation should be done in Tensorflow ***(Tensorflow.org, 2019)*** for several reasons. First of all, custom operations requires a low-level framework with direct and easy access to all intermediate data in a way that is not always straightforward with high-level frameworks.

Choosing from the low-level frameworks, Tensorflow is the strongest upcoming one being, or with potential to soon be, the machine learning standard framework.

Some MIDI framework should be used to read and write MIDI files.

- **Datasets**:

  The final dataset will be a set consisting of as many Mahler symphonies as can be found in MIDI format. Preferably, the model should be tried with simpler datasets before this and these should, with advantage, be well known and used in studied publications.

- **Evaluation**:

  The architecture should ideally be evaluated through loss and performance metrics as well as through its capability to generate music with idiomatic properties and quality. The former can be done by comparison to other models if the modelled properties are comparable to others. If not, reconstruction metrics and evaluation with respect to a baseline is possible. For qualitative evaluation, analysis of generated music should be done. A musical Turing test and the use of a listening panel might be interesting evaluation methods in some cases, but they don't necessarily say anything about the music generated since there is often too much uncertainty as to who is in the panel and how the music has been presented.

- **Source code**:

  Source code should be made publicly available online through for example GitHub.

- **Samples**:

  Samples should be made available online as well as written music in the results section.

4.2 MahlerNet

MahlerNet was implemented from scratch in Python with Tensorflow ***(Tensorflow.org, 2019)*** in the spring of 2019 along with its preprocessor, using Mido as MIDI framework to read and write MIDI files ***(Github.com, 2019)***. All the code along with instructions for running it are available at `https://github.com/fast-reflexes/MahlerNet` and listening samples are available at the MahlerNet website (`http://www.mahlernet.se`). MahlerNet is the result of the literature survey presented previously in this work and draws inspiration in its architecture mostly from MusicVAE, BALSTM and BachProp. The preprocessing draws inspiration mostly from PerformanceRNN and BachProp. Mahlernet is essentially a sequence to sequence network with a VAE modelling the latent space. A decoder is used to reconstruct offset, duration, pitch and instrument, basic components of written music as demanded by condition C.7, at each time step. Since it is a sequence to sequence network, the kind of modelling where a number of time steps are used to predict the next, as can be done with a plain RNN model, is out of the question and the decoder must produce the entire output from the latent code. However, the length of the sequence may be adjustable to encompass a bar, a quarter note or something else. In the remaining part of this work, this unit has been fixed to a bar, which is a variable measure given different time signatures. The choice of a bar is arbitrary but a natural one given that it is large enough to contain some musical patterns but still small enough not to contain sequences of events of such lengths that will early on cause potential problems in a neural network, thus a good initial guess that might be fine-tuned or further challenged in another work. A musical context preceding the music to model can be used as conditioning as well as additional inputs holding information about underlying metrical structure, currently sounding pitches and currently sounding instruments. After training, music can be sampled from MahlerNet and then reconstructed using the decoder and with the help of a conditioning context, the idea is that some long-term structure is enforced. MahlerNet is highly configurable and except for the regular choices that can be made with respect to layers and layer sizes, MahlerNet implements batch normalization, dropout and scheduled sampling. A VAE loss with free bits and a β parameter are further optional features. In the sections below on the different components of MahlerNet, full modelling using all features is considered to give as much insight as possible.

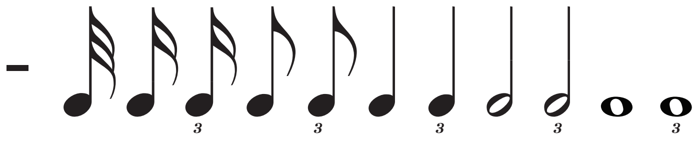
4.2.1 Modelled Properties

MahlerNet models polyphony in a sequential manner with the properties of offset to previous event, duration, pitch and instrument at every time step. It is possible to model any subset of the properties with the property of pitch being non-optional.

4.2.1.1 Offset and Duration

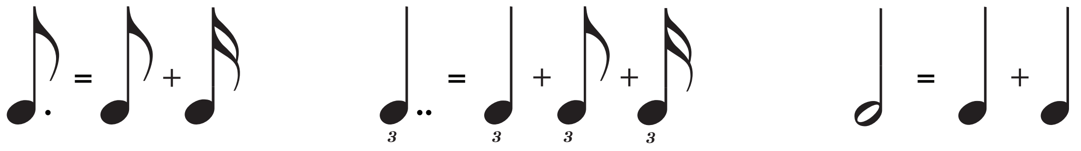
In this subsection, duration can refer both to the property duration (of an event) being modelled but also to the values that are used for both the duration and offset properties.

Offset is the property that makes time move forward and much like how time progresses in MIDI, it marks the amount of time, in some unit, that has passed since the previous event. Conversely, duration indicates for how long, in some unit, that an event will last for. Offset and duration are modelled in the same way with 60 classes when modelled in a local fashion. Optionally, offset and duration can also be modelled in a distributed fashion where only 11 classes are used. These 11 classes, along with the zero duration, are the base classes, consisting of both duplet and triplet durations. They represent a basic set of durations and no matter the representation, all other durations are constructed from them. The zero duration is not explicitly modelled with a distributed representation, where instead the lack of any active class represents this duration, whereas with a local representation, the zero duration is explicitly modelled. Durations not in the set of base classes are called compound durations, and they consist of merges of at least two *different* durations from the basic set, never including the zero duration.



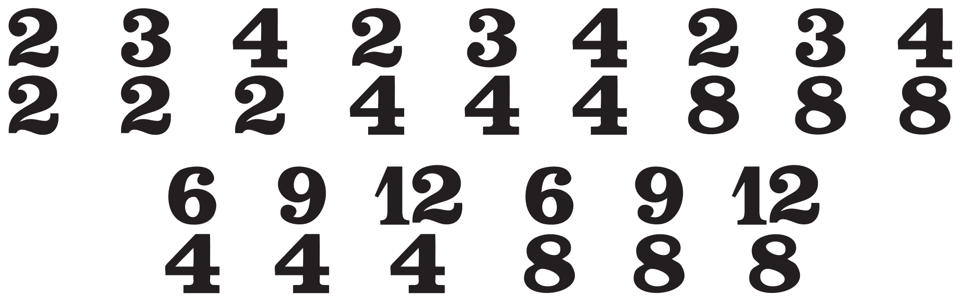
Graphic 4.1: The set of 12 basic durations with the zero duration first.

Among all durations, some are more idiomatic for music than others. The 12 base classes are very common but the more members of the base classes that are added together (in a way so that they can't be expressed as another duration) to construct a new compound duration, the more uncommon it tends to be. Compound durations consisting of two members from the base classes are also very common but three is more uncommon whereas four is very uncommon. Nevertheless, since the property of offset is also modelled with these durations, all sorts of compound durations occur. The offset and duration properties have their durations ordered in ascending order so that a distance measure between a prediction and the correct class can be induced to measure the magnitude of the prediction error.



Graphic 4.2: First two compound classes consisting of 2 and 3 durations from the basic set and then an example of a basic duration since even though it can be divided into other durations, the constituent durations are not different.

The shortest duration, except for the zero duration, is a 32nd note (demisemiquaver) and the longest is a 32nd note less than five quarters. This may seem like an arbitrary choice but has been chosen with respect to somewhat common situations with syncopes before a bar line that last for the full duration of the next bar. Longer notes than the maximum duration are divided into full bar notes with the first and the last subdurations being any appropriate class. Time signatures that use a triple meter are represented in duple meter with a triplet subdivision and the music modelled is limited to a few chosen time signatures.



Graphic 4.3: The accepted time signatures modelled by MahlerNet. Duple meters on top and triple meters below.

Any multiples of the numerator of the above time signatures are also accepted, should the denominator be the same. In these cases, each bar is simply broken down and interpreted as a number of bars of one of the accepted time signatures. During preprocessing, all different time signatures are parsed as ranges and all subsequent ranges form an input chunk of consecutive music, as seen from the perspective of MahlerNet. Parts of pieces with other time signatures than the above are discarded, resulting in an actual input song being divided into several.

4.2.1.2 Pitch

MahlerNet models 96 pitches ranging from one-indexed MIDI pitches 17 (incl.) to 112 (incl.) corresponding to the range from F0 to E8. The 32 remaining pitches were excluded due to that they hardly ever appear in any music.

4.2.1.3 Instrument

To try out MahlerNet and make processes simpler, a group of 23 different instruments have been modelled instead of all of the 128 available MIDI instruments, even though there is no inherent restriction in the architecture that forces this. The chosen instruments are capable of representing all the different classes of timbres and during preprocessing, the 128 MIDI instruments are mapped to one of these 23 instruments. Percussion, typically modelled on MIDI channel 10, is ignored as well as that specific channel.

Number	MIDI Instrument (number)	Number	MIDI Instrument (number)
1	Acoustic Grand piano (1)	13	Trombone (58)
2	Drawbar Organ (17)	14	Tuba (59)
3	Electric Guitar (clean) (28)	15	French Horn (61)
4	Electric Bass (finger) (34)	16	Brass Section (62)
5	Violin (41)	17	Tenor Sax (67)
6	Viola (42)	18	English Horn (70)
7	Cello (43)	19	Bassoon (71)
8	Contrabass (44)	20	Clarinet (72)
9	String Ensemble 1 (49)	21	Flute (74)
10	Timpani (48)	22	Synth Lead (square) (81)
11	Choir Aahs (53)	23	Pad 1 (new age) (89)
12	Trumpet (57)		

Table 4.1: MIDI instruments that MahlerNet models and to which all other encountered instruments are mapped. MIDI instrument number in parenthesis.

4.2.2 Preprocessor

The preprocessor works on the representation of MIDI files as returned by Mido where the tracks of a MIDI file are arrays of Mido events. The preprocessor is a pipeline where several functions processes the events of a file iteratively, each time adding to or normalizing some aspect of it. Several settings can be done in the preprocessor resulting in representations of different size. For example, the range of modelled pitches can be set as well as which basic durations to mark in the underlying meter via the beats vector. The output of the preprocessor for a given MIDI file is a tuple containing meta information about the preprocessing as well as a series of data vectors consisting of all the events in the file.

4.2.2.1 The MIDI Format

The MIDI format was proposed in 1981 and is an incredibly rich format that allows for a detailed representation of music of all forms. MIDI can be recorded from electronic instruments and thus capture live music performance as good as it can store written music in symbolic form generated by music engraving software. Albeit, there is not a one-to-one mapping between MIDI and written music.

A MIDI file is a sequence of data chunks with one header chunk and one or several track chunks, containing data for a track ***(ntu.edu.tw, 2019)***. MIDI files come in three flavours where type 0 holds a single track, type 1 holds multiple tracks that are to be played simultaneously and type 2 holds multiple tracks to be played independently. Type 1 is typically most common when listening to general music recorded as MIDI.

A track is an ordered series of events grouped together in some way. It typically represents an instrument or a voice (in polyphony) but there is no restriction against placing all events in the same track in a MIDI file. However, the track concept makes it easier to visualize the music in sequencers for example. Each track holds many type of events and meta events. Turning on or off a note (pitch) are two types of events but also a change in tempo, change in key signature or selecting a new program (instrument) for a given channel. There is no restriction on the number of tracks in a MIDI files but actual sounds go out through the concept of channels, which are 16 to the number in standard MIDI. Different tracks can start and end output on any channel.

MIDI events use relative timing and given that time is 0 at the beginning of a MIDI file, every track, independently, uses an offset for each event to distinguish how many ticks have passed since the last event in the track. Each MIDI file has a ticks per beat indication which indicates how many ticks that corresponds to a quarter note. The concept of tempo is also used, and can be changed during the course of a MIDI file, and refers to the number of microseconds per quarter note. The default value for ticks per beat is 120 and the default for tempo is 50000 microseconds per beat. These two values form a conversion system between actual time and note values which must be used in order to convert MIDI music to both sounding and written music. Needless to say, with the default values used here, MIDI has the ability to express micro-timing on a high level and as previously stated, the length of a MIDI event does not necessarily correspond to a feasible written duration.

Worth to point out is that typically, both channels and tracks provide information about music grouped somehow in the MIDI file. Channels because a channel only plays one instrument at a time and all events being output on the channel thus correspond to the same instrument being used which is equal to a grouping of the music played. Conversely, tracks, as stated, are also typically either the same instrument groups or subdivisions of those in voices.

The most intuitive format of how a MIDI file is played is acquired by merging all tracks and then updating all offset counts accordingly.

4.2.2.2 Tokenization

The initial step of the preprocessing is the tokenization which in turn is divided into two parts. The first part is called *streaming* because it streams all the events from the input file, track by track, into a list where each event is wrapped with additional information about what track it comes from and with an absolute timestamp, as opposed to the relative timestamp used in MIDI. All events except events turning notes on and off, tempos, time signatures and program changes (instrument changes) are discarded. The next step is the actual *tokenization* in which the list from the previous step, ordered in ascending order with respect to the absolute starting time of the events, is processed anew. The purpose now is to merge events starting a note with the corresponding end event so that a duration in ticks can be calculated for each note. During this step, tokens within the same track are prolonged and shortened using a heuristic with the purpose to normalize MIDI files resulting from recordings. The idea is that, for example, when recording on a piano, the player is likely to both start and end notes both early and late at times, and by looking at previous and next notes in the same track, one might conclude whether to shorten or prolong a note. For example, if two relatively long notes are played after each other and there is an overlap corresponding to only a small fraction of this duration, the first note was probably held too long. An analogue reasoning goes for notes being released slightly early. During tokenization, the list indices of encountered time signatures are stored. Each output event is associated with an instrument and encountered instrument changes are used to govern this. Current tempo is also stored for each token. After tokenization, a MIDI file is a list of tokens where each token has a duration in ticks, an instrument, an absolute starting time with respect to the start of the file and a tempo. The list is sorted in ascending order with respect to absolute starting time and pitch height. At this point, a note is represented by one event, and not two indicating start and end.

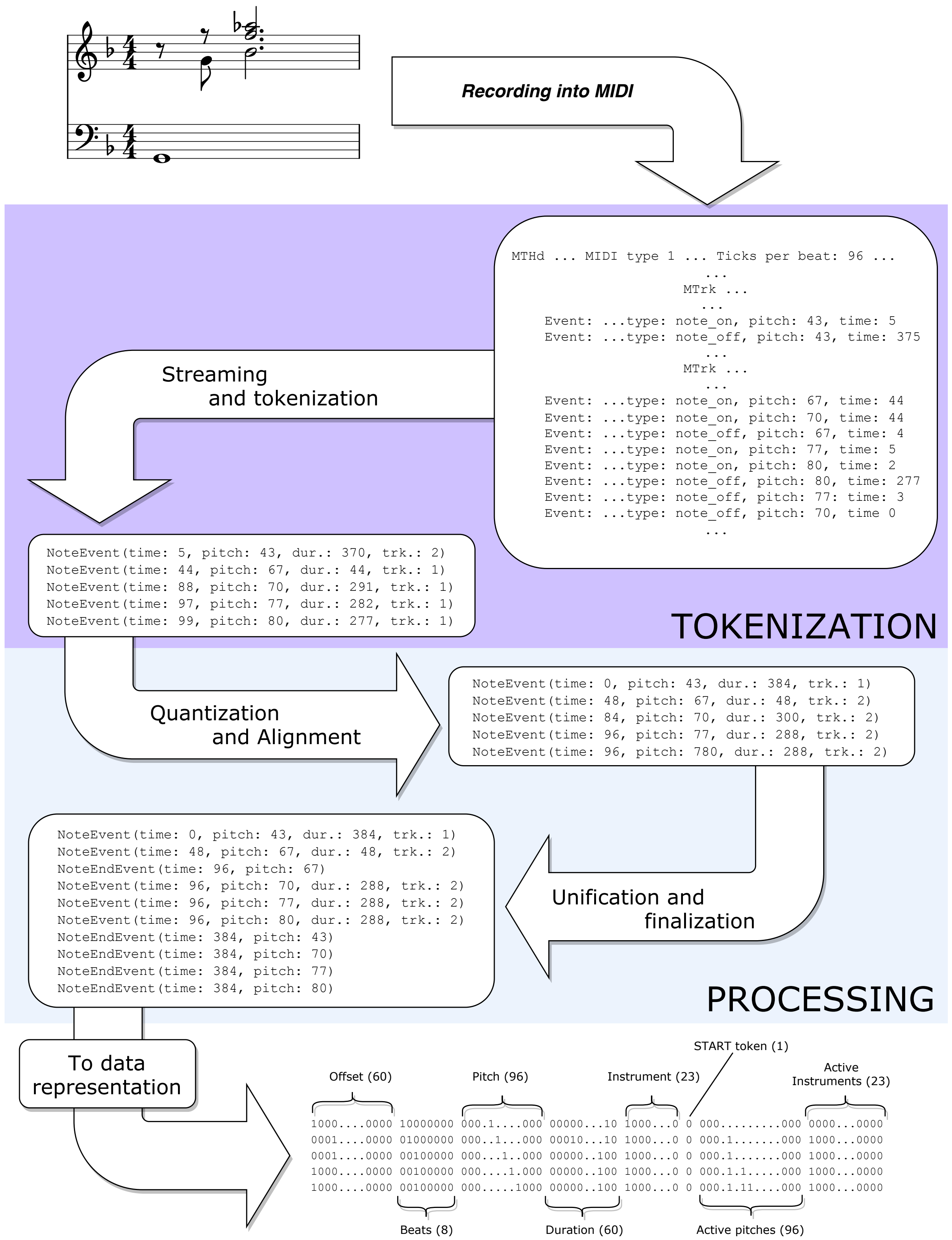
4.2.2.3 Processing

From this part of the preprocessing, the list of event tokens is chunked into ranges of tokens where each range corresponds to a time signature. In all subsequent steps, these ranges are processed individually but if they are uninterrupted, they are merged in the data representation. Interruptions occur when some illegal time signature, e.g. a time signature that can't be handled by the preprocessor, is encountered. This effectively creates an interruption in the MIDI file and the subsequent part, if any, is considered an entirely new piece of music in the data representation. The term *chunk* is used to denote such a series of one or more contiguous ranges and a *range* is simply a series of events not interrupted by a new time signature event.

The *processing* step is divided into three substeps. The first step is called *quantization and alignment* and represents the most elaborate step of the whole preprocessing procedure. It starts out by finding candidates among the duration classes for the duration (in ticks) of an event. Depending on input duration in ticks, a number of candidates from the overall list of classes is returned, both longer and shorter than the current duration of the event. A heuristic algorithm is then used to calculate a score for each candidate that is based on the difference between the current duration and the duration candidate. The score is also based on the difference between the current starting time of the event and the time when the event would preferably have to start given the candidate. The idea is to analyze the partial components (basic durations) of the candidate and align the largest or second largest partial in a way that is acceptable relative to the underlying meter. This may force the starting time of the event to be moved which contributes to the score. The heuristic uses information about both tempo and track to determine which of a set of predefined overall lists that the candidates are chosen from; in a fast tempo, 32nd notes are excluded and when the previous note was a triplet, some of the duplet durations are excluded, for example. The lowest score wins and after this first step of processing, each event has been aligned and quantized and represents its duration with one or several durations classes instead of a duration in ticks. Notes longer than the maximum duration are divided into as many durations as is necessary and are taken care of by picking candidates for the durations arising after removing as many full bars as is possible. MahlerNet thus divides long notes into several shorter notes.

The second step of processing is called *unification* and here, notes are unified in the sense that vertical groups of pitches are aligned and prolonged or shortened depending on the majority vote of the group. Track info are used here as well to determine which events to unify.

Finally, in the third part of processing, events are split into several sequential events whenever an event duration cannot be expressed with one duration class only. Also, each event is anew divided into two and an end event is placed out chronologically for all events. Nonetheless, the starting events still hold all vital information and the end events are simply there to facilitate the use of active pitches and instruments in the data representation. After the processing step, there is a one to one mapping between the list of tokens and their data representation.



```
MTHd ... MIDI type 1 ... Ticks per beat: 96 ...
                        ...
                      MTrk ...
                        ...
    Event: ...type: note_on, pitch: 43, time: 5
    Event: ...type: note_off, pitch: 43, time: 375
                        ...
                      MTrk ...
                        ...
    Event: ...type: note_on, pitch: 67, time: 44
    Event: ...type: note_on, pitch: 70, time: 44
    Event: ...type: note_off, pitch: 67, time: 4
    Event: ...type: note_on, pitch: 77, time: 5
    Event: ...type: note_on, pitch: 80, time: 2
    Event: ...type: note_off, pitch: 80, time: 277
    Event: ...type: note_off, pitch: 77: time: 3
    Event: ...type: note_off, pitch: 70, time 0
                        ...
```

```
NoteEvent(time: 5, pitch: 43, dur.: 370, trk.: 2)
NoteEvent(time: 44, pitch: 67, dur.: 44, trk.: 1)
NoteEvent(time: 88, pitch: 70, dur.: 291, trk.: 1)
NoteEvent(time: 97, pitch: 77, dur.: 282, trk.: 1)
NoteEvent(time: 99, pitch: 80, dur.: 277, trk.: 1)
```

```
NoteEvent(time: 0, pitch: 43, dur.: 384, trk.: 1)
NoteEvent(time: 48, pitch: 67, dur.: 48, trk.: 2)
NoteEvent(time: 84, pitch: 70, dur.: 300, trk.: 2)
NoteEvent(time: 96, pitch: 77, dur.: 288, trk.: 2)
NoteEvent(time: 96, pitch: 780, dur.: 288, trk.: 2)
```

```
NoteEvent(time: 0, pitch: 43, dur.: 384, trk.: 1)
NoteEvent(time: 48, pitch: 67, dur.: 48, trk.: 2)
NoteEndEvent(time: 96, pitch: 67)
NoteEvent(time: 96, pitch: 70, dur.: 288, trk.: 2)
NoteEvent(time: 96, pitch: 77, dur.: 288, trk.: 2)
NoteEvent(time: 96, pitch: 80, dur.: 288, trk.: 2)
NoteEndEvent(time: 384, pitch: 43)
NoteEndEvent(time: 384, pitch: 70)
NoteEndEvent(time: 384, pitch: 77)
NoteEndEvent(time: 384, pitch: 80)
```

```
1000....0000 10000000 000.1....000 00000...10 1000...0 0 000.........000 0000...0000
0001....0000 01000000 000..1...000 00010...10 1000...0 0 000.1.......000 1000...0000
0001....0000 00100000 000...1..000 00000..100 1000...0 0 000.1.......000 1000...0000
1000....0000 00100000 000....1.000 00000..100 1000...0 0 000.1.1.....000 1000...0000
1000....0000 00100000 000.....1000 00000..100 1000...0 0 000.1.11....000 1000...0000
```

Graphic 4.4: Schematic showing the different preprocessing steps and a simple illustration of how data is transformed. The input to the preprocessor is MIDI and the recording to MIDI is not done as a preprocessing step since available MIDI data has been used. This step is nonetheless included to emphasize that there is not a one-to-one correspondence between written music and MIDI, hence the necessity for elaborate preprocessing.

4.2.2.4 Data Representation

This is the final step before the MIDI file has arrived to its data representation. This step is rather simple and processes the tokens, and for each start token, a vector representing a new time step is added to the output holding the offset, duration, pitch and instrument of the given event. For each event is also added a beat vector showing if the current starting time is aligned on an eighth note and if so, where in the bar this eighth note is. Ending events are used to keep track of the active pitches and active instruments maps that are supplied as well to each time step, optionally used to make it easier during training to predict the next pitch and instrument. All in all the data representation used for the experiments to come has 367 dimensions event though the preprocessor, as said, allows for options that may change this number. The effective number of dimensions may also change depending on what options are used in training. A single dimension holds a START token, set to 1 only for the first time step of contexts that include units of music from before a piece have started, i.e. when the music to reconstruct is the first bar of a piece.

4.2.3 Encoder

The encoder part of MahlerNet has two encoders in it capable of encoding one unit of music each. Both encoders are bidirectional RNNs with a variety of cell options (both LSTM and GRU). Two different variations on the bidirectional RNN, governing how the bidirectional information is passed between layers can be chosen from as well. The *input encoder* encodes the unit of input music that is to be reconstructed by the decoder during training. The *context encoder* encodes the unit of music preceding the input unit, used to condition both the VAE and the decoder during training. During inference, the context is used to condition the decoder that reconstructs from a latent vector sampled from the VAE.

After encoding, the forward and backward final states of the encoders are concatenated and passed to the VAE.

4.2.4 Variational Autoencoder

The *variational autoencoder* (VAE) uses the summary from the encoder component and passes it through two feedforward layers to yield a parameterization for the latent space in terms of mean and logarithmic variance. The former has no activation whereas the second feedforward layer uses a softplus activation function as is common for the variance part of the VAE. The reparameterization trick is used whereby a sample from a standard Gaussian is used as a constant to multiply the variance with to yield a sample from a Gaussian distribution with the previous mean and variance without breaking the ability to propagate gradients through the VAE.

The sampled latent vector can be used to condition the decoder in multiple ways. For example, it can be used to generate different or similar projections for multiple layers and starting state and also be used as conditioning during decoding in every time step. For the experiments in this work, all initial states of all layers of the decoder are initialized with the same projection of the latent vector which is also passed as conditioning in every time step during decoding.

4.2.5 Decoder

The decoder uses the latent vector sampled from the VAE and produces its initial states by running the latent vector through a linear layer with tanh activation, as is done in the MusicVAE. As described above, the same projection is used as conditioning during the entire decoding process. The recurrent part of the decoder is modelled either by a "regular" decoder that consumes the feature vector as a whole each time step, or a BALSTM that divides each batch sequence into 96 sequences individual for each pitch. The "regular" decoder is from hereon referred to as an RNN decoder and it can be used with different cells, such as GRU or LSTM cells.

4.2.5.1 RNN

An RNN decoder models the full range of pitches, along with other features, as one input vector and may thus not, at least intuitively, account for pitch invariance since pitch patterns shifted a number of steps will look different to the RNN and yield different output. However, a regular RNN is simple to work with and lends itself to use with scheduled sampling in an easy way. Unfortunately, the extra conditionings available with beats and active pitches and instruments depends on the original sequence (correct sequence) and can thus not be used with scheduled sampling when using the regular RNN decoder.

4.2.5.2 BALSTM

The BALSTM uses the notion of time axis and note axis, hence its name BiAxial LSTM. The original BALSTM models all pitches occurring at the same time in a single time step, but independently in one LSTM per pitch. The output of the time axis module is a binary node with sigmoid activation (or softmax activation over two classes) determining if that note should be on or not. Up to this point, all notes in a time step are independent of each other. The note-axis then processes the notes from lowest to highest, again recurrently but now in the note-axis, to yield the final output probability at which time the notes in a time step are no longer independent.

There are two important concerns to address with respect to MahlerNet at this point. First of all, since MahlerNet does not model several notes in a time step, the role of the note-axis module is unclear. Furthermore, the binary output nodes of both the time-axis and note-axis modules allow for several notes to be sampled as active during a single time step whereas in MahlerNet, we must only allow one pitch to be active in a given event. To address these issues, a modification of the BALSTM is introduced whereby the note-axis is removed and the output of the individual pitch RNNs are concatenated to form the input to a softmax layer that yields a distribution over all the possible pitches. Since all pitches are modelled individually and only with a window spanning its close vicinity, the transposition invariant property holds even though the biaxial property is really reduced to a uniaxial one. Nonetheless, the fact that this design stems from the BALSTM allows for it to be called a variant of the BALSTM.

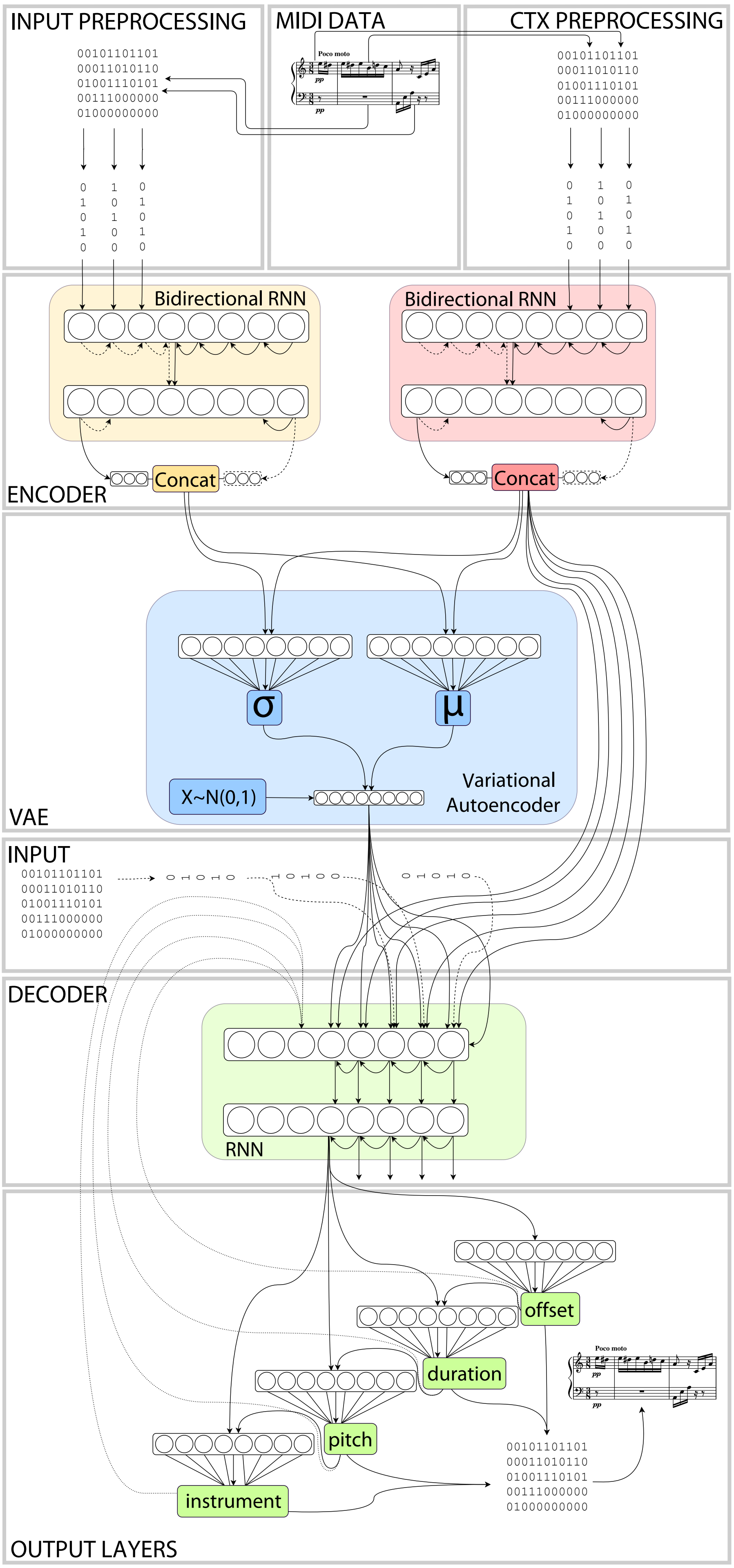
Due to previous observations, the inputs to the modified BALSTM with respect to the original, are also changed. The original BALSTM receives a vector of last time step activities in the two-octave vicinity of the pitch to be modelled along with an octave vector of activities across invariant pitches over the whole range. This setup does not seem to be optimal considering that the last time step contains only one active pitch. Instead, we serve the modified BALSTM with the vicinity vector only and also, optionally, a vicinity vector from the active pitches map which indicates which pitches are currently active.

As per recommendation in DeepJ, the vicinity vector is also replaced with a one-dimensional convolution over the vicinity where 64 filters are used. In DeepJ and the original BALSTM, the surrounding range of an octave in both directions is used as vicinity whereas Mahlernet uses two octaves plus 1 note, placing the pitch to model in a range where it has access to the same pitch class one octave up and one octave down. In places of the register where the range doesn't extend further, the input is zero-padded, as is also done in the original BALSTM network. The MIDI pitch number to be modelled by a specific instance of the pitch RNN is also included as a one-hot input. In DeepJ and BALSTM, this input is a discrete number instead.

One downside of using the BALSTM is that it makes scheduled sampling incredibly complicated since sampling from a single pitch RNN instance would not make sense since all instances from a given time step must be used for sampling to take place. This implies that groups of pitch RNNs modelling all pitches in a time step for a batch sequence must be sampled simultaneously which requires an elaborate customization. Scheduled sampling is thus not used with the BALSTM in the experiments to come.

4.2.6 Output Layers

The output from the recurrent part of the decoder in MahlerNet is run through softmax output layers to acquire distributions over each of the modelled properties which are modelled at every time step. This is a step away from the representation in PerformanceRNN where each time step only models one property and the network has to choose between progressing time, turning pitches on or off and adjusting the dynamics. The advantage of the latter representation is that zero offsets need never be modelled which is a must in MahlerNet whenever no time passes between events. As in BachProp, offset is modelled first, then duration which is also conditioned on the current time step offset along with beats. Conditioned on all of the above, pitch is then predicted and then finally instrument. Modelling properties separately (as is sometimes the case when modelling for example pitch and duration only) is not an option since the properties clearly correlate; a short duration allows for a pitch much more dissonant to the current key than does a long duration.



Graphic 4.5: MahlerNet in its entirety when all its conditioning functionality is used. At the top, the preprocessor generator generates training data for the input and context RNNs. The context part of the network can be turned off, in which case all the connections arising from the right bidirectional RNN in the figures are left out. The contextual RNN is fed with content appearing in the source data one unit (here a bar of music) before the input. That is, if the input consists of bar 5 out of a piece of music, the context gets fed with bar 4. The input part is what MahlerNet is supposed to reconstruct during training. When the input is fed with the first unit of music, the context is fed with a special start token only. During training, the input is also fed (shifted one time step starting with an all-zeros time step) to the decoder whereas during inference, or when using scheduled sampling, the next time step is supplied from the output of the output layers. Output layers, except pitch, may also be turned off to model a subset of the properties. Inputs are always merged by concatenation and never by addition.

Two other mechanisms are also in place to further guide the outputs of MahlerNet. On rare occasions, the preprocessor must output empty time steps that only makes time progress. This happens for example when there is a need to align a triplet subdivision to the next duplet subdivision or vice versa. Evidently, in these situations, the offset property will be non-zero, since there is otherwise no point to output an empty time step. With this information, the duration logits are masked so that whenever the offset is zero, the zero class of the durations logits are set to a very large negative number, to make it virtually impossible to predict the zero duration class. The same sort of masking takes place when outputting pitch, effectively setting all pitch logits corresponding to pitches below the one of the last time step to large negative numbers whenever the offset is zero. Alas, events are ordered when output from the preprocessor in ascending order both on starting time and on pitch and so a lower pitch can start after a higher pitch if no time passes in between.

4.3 File Organization

The implementation of MahlerNet contain several files and a certain file organization is in place both with respect to training data and the actual program files.

The following files together make out the implementation done in this thesis:

```
<MAHLERNET ROOT>
       |
       |- MahlerNet.py
       |- RunMahlerNet.py
       |- MidiPreprocessor.py
       |- ProcessFolder.py
       |- DataProcessor.py
       |- DurationFactory.py
       |- Events.py
       |- Instruments.py
       |- display_stats.py
       |- utilities.py
```

The actual network is implemented in `MahlerNet.py` whereas the file `RunMahlerNet.py` manages training and generation with several preconfigured alternatives. `MidiPreprocessor.py` is the preprocessor for MIDI and it is used by `DataProcessor.py` that outlines several functions that batch processes a group of files. For example, it can turn a group of MIDI files into data representation, output MIDI files from a data representation and set up generators from single or multiple files stored in data representation format. `ProcessFolder.py` is a convenience class on top of `DataProcessor.py` that manages calls and command line arguments in a simple way. `DurationFactory.py`, `Events.py` and `Instruments.py` contain classes used for processing by `MidiPreprocessor.py`. Finally, `display_stats.py` consumes command line arguments and displays statistics about one or several training sessions using the records file created after every training session (if this option is turned on). `utilities.py` contains a few general functions connected to formatting.

The input data is expected in a specific format:

```
<DATASET ROOT>
      |
      |- input
      |- data
      |- seq
      |- midi
      |- runs -------|
         <UNIQUE TRAINING SESSION ID>
         ...
         <UNIQUE TRAINING SESSION ID>
                     |
                     |- params.txt
                     |- all_params.txt
                     |- commands.txt
                     |- graphs
                     |- trained_models
                     |- records
                     |- generated
```

The `input` folder is expected to have all the dataset MIDI files in a structure that is flat or hierarchical with subdirectories. This doesn't matter since `DataProcessor.py` will gather all the files it can find in the directory subtree. After conversion to data representation, all the files (that were successfully turned into data representation) are represented in the `data` folder as well. In this folder, all the files lie separately but in data representation. For training, the `seq` folder is constructed that contains sequences of the desired length (in the experiments, we always assume that a unit of music is one bar but MahlerNet supports other choices as well. In the `seq` folder, files are created that hold sequences of the same length, to make training more efficient. If desirable, one could turn the files in the `input` folder back into MIDI representation to have a glimpse at what they look and sound like after conversion to data representation, in which case they are stored in the `midi` folder.

Every training session requires a unique name which is used to store data about the session in the `runs` folder. In this directory, all the training sessions for the given dataset reside with unique names. The structure in such a training session folder is the same for all sessions and the META files `params.txt`, `all_params.txt` and `commands.txt` contain the input parameters used to run the session, all of the parameters (including the internal parameters that cannot be modified via an input configuration) used to run the session and finally the command used to initialize the session. In the `graphs` subfolder, graphs saved from the session can be found (if this option is turned on) and in the `records` subfolder MahlerNet saves statistics about the training session in a file named `records`. `trained_models` contains models saved during checkpoints while training and if anything is generated, during or after training, it is saved in the `generated` subfolder.

4.4 Running MahlerNet

Most of the executables have an argument parser that allows for help and information about arguments via the `--help` flag. Nonetheless, a few typical use case scenarios are presented below. Running MahlerNet itself requires an external configuration file that must contain information vital to building the network prior to training or running inference tasks. An example configuration file can be found at the MahlerNet Github page at `https://github.com/fast-reflexes/MahlerNet`.

To convert all the MIDI files in the folder `<FOLDER>/input` to data representation placed in `<FOLDER>/data` run

```
python ProcessFolder.py <FOLDER> -d
```

Similarly, to convert to data representation and training sequences (using a bar as a unit of music) and convert the data representation back into MIDI to store in the folder `<FOLDER>/midi`, run

```
python ProcessFolder.py <FOLDER> -dsm
```

To run a training session named `<ID>` with dataset root folder `<DS_ROOT>` and config `<CONFIG>`.

```
python RunMahlerNet.py train <ID> --root <DS_ROOT> --config <CONFIG>
```

During training, both the average time step loss as well as average accuracy metrics are displayed. Since the model in general must choose one class for each property every time step, and there is one correct class for every property at every time step, the precision and recall metrics are always the same. The accuracy shown during training is, in fact, exactly the precision or recall metric.

To display statistics about the reconstruction percentages for individual properties after running a training session named `<ID>` on the dataset located in the `<DS_ROOT>` folder, run

```
python display_stats.py p <PLOT_NAME> <DS_ROOT>/runs/<ID>
```

The `PLOT_NAME` is just a name to print in the diagram legend and any number of pairs `<PLOT_NAME> <PATH_TO_DATASET_FOLDER/runs/ID>`, where `ID` is a valid training session identifier for that dataset can be used to display multiple statistics at once.

To reconstruct the first bar in a given file `<INPUT_FILE>` available under that name in the `data` folder of the dataset root in question, run

```
python RunMahlerNet.py generate <ID> --root <DS_ROOT> --type recon --file
    <INPUT_FILE> --units 0cz --model <CHECKPOINT_NAME>
```

The sample is placed in the `generated` folder previously mentioned and `<CHECKPOINT_NAME>` has to be the name of a checkpoint saved in the `trained_models` directory of the `runs` folder named `<ID>` situated in the dataset folder `<DS_ROOT>`.

Conversely, run

```
python RunMahlerNet.py generate <ID> --root <DS_ROOT> --type pred --length
    10 --meter 3 --file <INPUT_FILE> --units 0cz --model <CHECKPOINT_NAME>
```

to sample a 10-bar sample from the same model as above, in 3/4 meter seeded by a context with only a START token (since we are reconstructing the first bar of a file) and an initial latent vector generated from the first bar in the given input song.

To measure performance on the reconstruction of an entire dataset with teacher forcing, run

```
python RunMahlerNet.py generate <ID> --root <DS_ROOT> --type n_recon --model
    <CHECKPOINT_NAME> --use_teacher_forcing
```

More info about running MahlerNet can be found on the project Github page `https://github.com/fast-reflexes/MahlerNet`.

4.5 Experiments

MahlerNet was continuously developed and tested for several months. The experiments were divided into three parts with increasing difficulty. In the last part, Mahler symphonies were modelled. The first part focuses on modelling pitch only with the ESSEN and SESSION datasets whereas the middle part uses the intermediate datasets NOTTINGHAM, PIANOMIDI and MUSEDATA. In the last part, MAHLER is used, which is a dataset with most of the Mahler symphonies gathered specifically for this work. More info about the datasets is presented in section 5 and in Appendix G.

5 RESULTS

MahlerNet was tested in three experiments with different circumstances. The first two experiments have the same main flow but the second experiment is much more elaborate and explores all features of MahlerNet. In the third experiment, the ordeal of modelling Mahler symphonies is taken on. In the first experiment, only pitch is modelled with basic settings whereas in the second and third experiment, all properties modelled by MahlerNet are included in the test. Training was done on a GeForce GTX1080Ti GPU. All samples are sampled at softmax temperatures of 1.0 if nothing else is said. The presented samples are neither the result of extreme cherry-picking nor the first samples to be generated but typically the best from about 2-4 attempts per shown example and thus moderately representative of what the models can actually accomplish. No elaborate postprocessing has taken place on any sample (except direct conversion with the preprocessor to MIDI) and all samples can be heard at the MahlerNet website (`http://www.mahlernet.se`) using the sample numbers that are stated whenever samples occur. A few samples are shown in this section but due to lack of space, too many couldn't be fitted. It is advised to follow along in the samples on the website while reading.

Wherever accuracy or reconstruction is mentioned, the percentage refers to precision, or recall since given the problem context these two metrics are the same. This in contrast to another interpretation of accuracy which takes true negatives into account as well, generally resulting in a misleading metric stationary around 100%.

5.1 Experiment 1: Modelling Pitch

In this experiment, MahlerNet was used both with and without the contextual input used to condition the VAE and the decoder to create longer coherence, and was subsequently sometimes run as a plain sequence to sequence model only using a VAE. No dropout or any other regularization was used except for the VAE loss which was used in its standard form, and as such, constituting a sort of regularization.

Encoder	**VAE**	**Decoder**	**Learning rate**
2*128	64	2*128	0.001

Table 5.1: Basic setup for experiment 1.

In this experiment, the ESSEN and SESSION datasets were used. Both of these mainly contain monophonic music and even though one could model pitch sequences only even with polyphonic pieces, it is much more intuitive to do so with melodies. The main purpose of the experiment with pitch only is to make sure that MahlerNet works as intended and to evaluate the BALSTM decoder against the regular RNN decoder. First, short runs with basic setups are used to determine a single BALSTM setup as well as a plain RNN decoder setup, which are then used in more elaborate tests with scheduled sampling and eventually with context.

Dataset	**samples**	**max seq. length**	**avg. seq. length**
ESSEN	116046	15	3.67
SESSION	1409937	24	6.03

Table 5.2: Information about the datasets used in experiment 1.

5.1.1 Short Runs (VAE Only)

Four regular RNN decoders were tried against four BALSTM decoders. The optimizers Adam and RMSProp were used as they had proven promising during prototyping and both LSTM units and GRU units were tested. Each model used a batch size of 256 and was run for 6 epochs on both the SESSION and the ESSEN datasets, the former resulting in 33048 gradient updates and 2724 for the latter. All data was used for training and the concept of overfitting was ignored, both due to the nature of a sequence to sequence model with no desire to generalize and also to the early phase of the experiments where confirmed overfitting is desirable and confirms the models expressibility and correctness of programming.

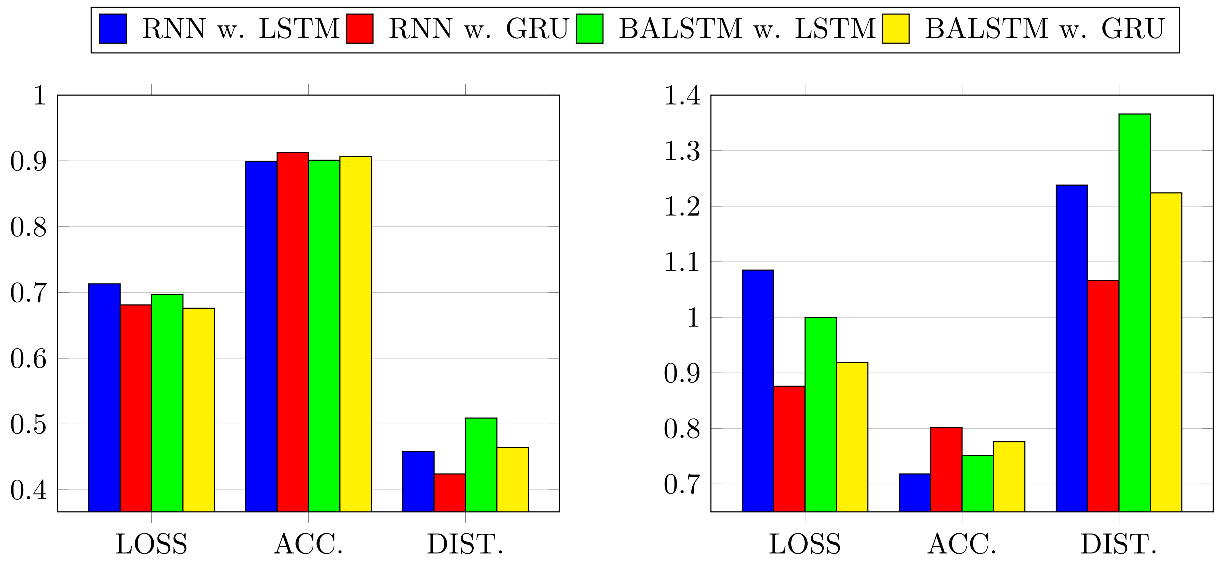


Figure 5.1: Comparison between an RNN decoder and a BALSTM decoder. Accuracy is measured on training set with teacher forcing and distance is the average number of classes away from the ground truth on each time step. The left plot is for the SESSION dataset and the right for the ESSEN dataset.

The diagrams show the best given setup (that is, the best result from using Adam and RMSProp with the setup in question) for the SESSION dataset to the left, and the ESSEN dataset to the right. Worth to point out is that the ESSEN dataset is much smaller so the number of gradient updates were much lower in that case. From the diagrams, it can be noticed that the GRU units perform better in general and that the RNN and BALSTM decoders are similar in performance, with the exception that the BALSTM decoder tends to be slightly farther off when it makes a bad prediction.

5.1.2 Long Runs (VAE Only)

Picking the best RNN and BALSTM setups from the last section, each model was tried on the same datasets resulting in 12 epochs for the SESSION dataset (66096 steps) and 36 for the ESSEN dataset (16344 steps). All the best models use GRU units and in all cases except one, they also use the RMSProp optimizer. The exception is the BALSTM for the SESSION dataset that uses Adam. A third setup with the RNN decoder using scheduled sampling with an inverse sigmoid schedule and a rate of 1000 was also employed now. The schedule is the same as is used in the MusicVAE. The batch size of 256 from last section is still in use and for the same reasons as earlier declared, all data was used for training and overfitting was ignored.

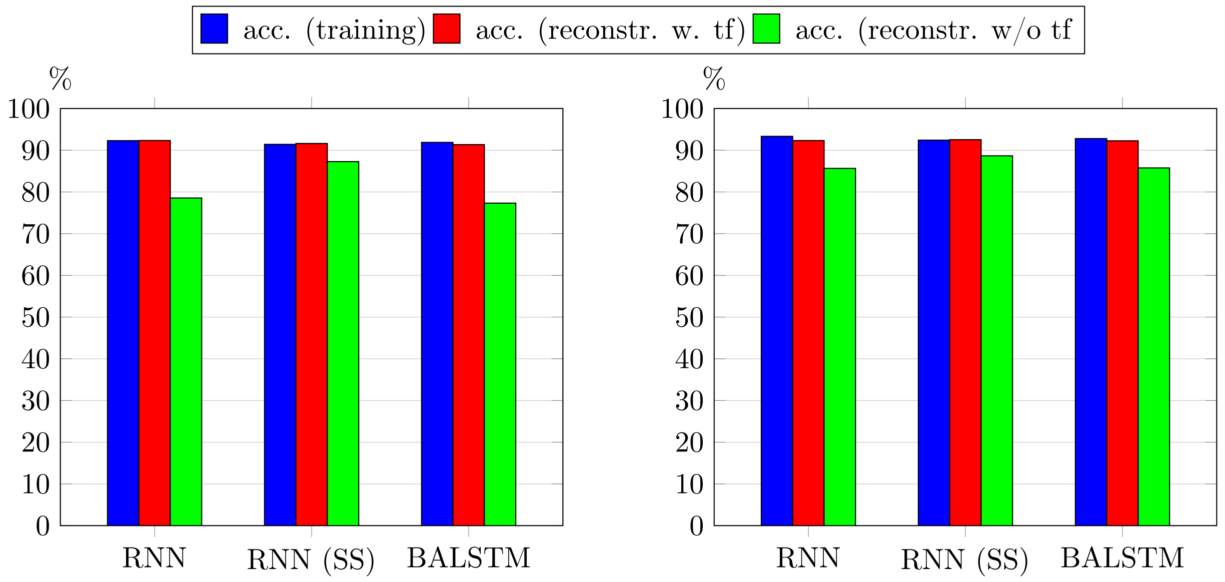


Figure 5.2: Comparison in reconstruction ability between an RNN decoders with and without scheduled sampling and a BALSTM decoder. The left plot is for the SESSION dataset and the right for the ESSEN dataset.

Accuracy was measured with respect to the length of the target sequences and thus, if, without teacher forcing, models generated longer sequences than the target sequence, only the first applicable output tokens were taken into account. Conversely, if the output sequence was shorter, all the remaining tokens in the target sequence were assumed to be bad predictions and contribute negatively to the accuracy statistics. Reconstruction outside of training was measured on 200 000 samples from the SESSION dataset which consists of 1409937 samples and on the full dataset for ESSEN dataset consisting of 116046 training samples.

5.1.2.1 BALSTM Transpositional Invariance

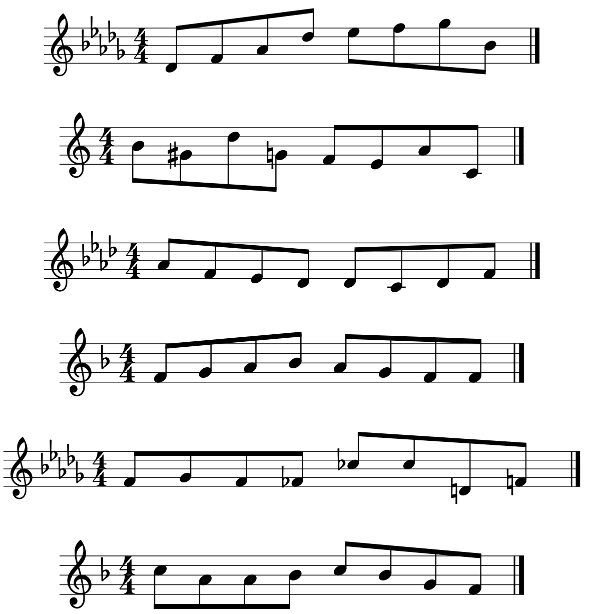
The SESSION dataset was transposed six steps up and both the BALSTM and RNN decoder from last section was used to reconstruct 200000 samples from the full set of 1409937 samples.

decoder	w. tf	w/o tf
RNN	87.53%	73.82%
BALSTM	86.73%	70.20%

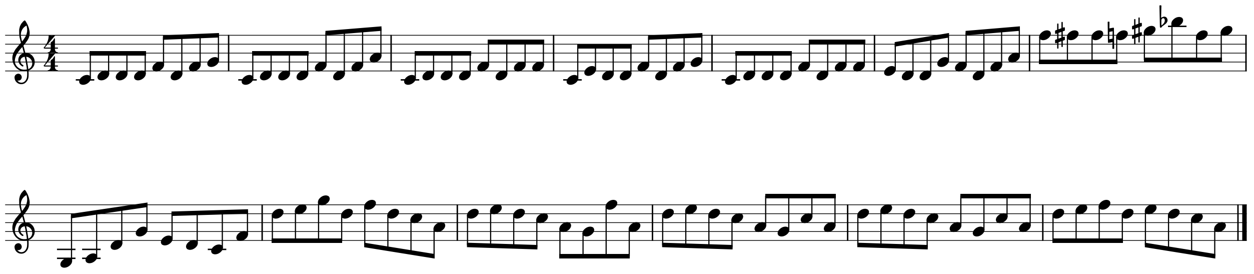
Table 5.3: Reconstruction accuracy on a transposed version of the SESSION dataset with and without teacher forcing.

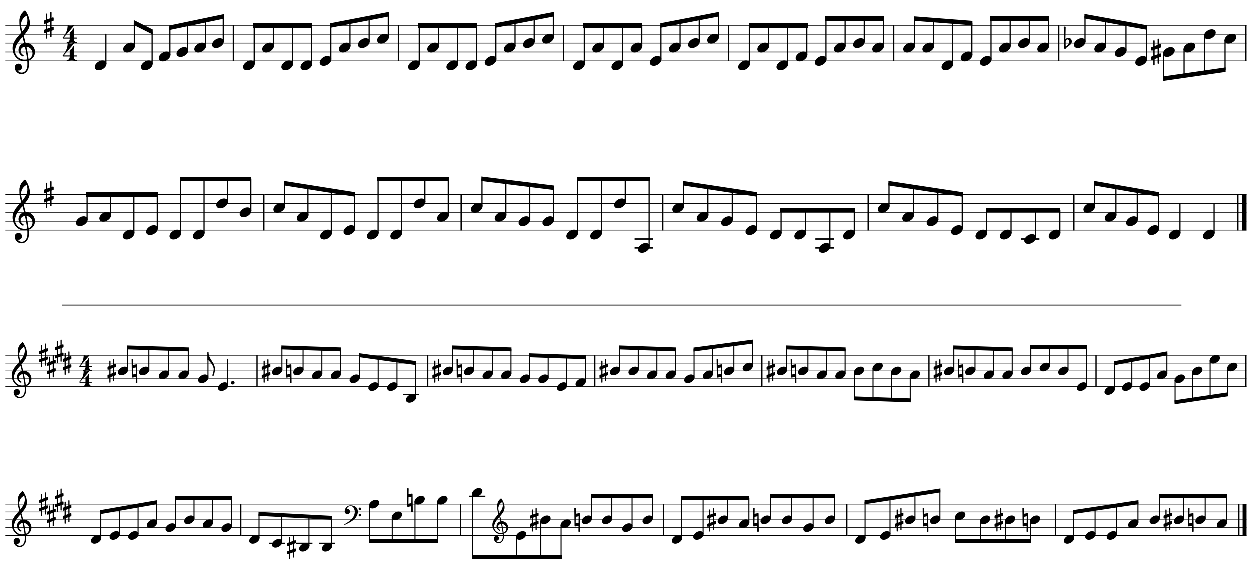
5.1.2.2 VAE Latent Space

The surface of the latent space on the models from the previous sections were explored by means of sampling and interpolations. 1-bar random samples were drawn from the three previous models for both the SESSION and the ESSEN datasets. Interpolations were done with SLERP, as proposed in MusicVAE, meaning that spherical interpolation were used instead of linear. Interpolations were done in 10 steps which results in a total of 11 steps including the start point. Examples show 13 bars where the first and the last bars are the original input bars, simply copied in, that are interpolated between. Since no duration or offset is modelled at this point, a static value was chosen for these properties (eighth note for both) and the model then yielded as many predictions as necessary to output a full bar.



Example 5.1: Samples 1-10, 1-11, 1-12, 1-13, 1-14 and 1-15. Sampled bars from MahlerNet with RNN decoder, RNN decoder trained with scheduled sampling and BALSTM decoder, first three with models trained on the SESSION dataset and then the bottom three with models trained on the ESSEN dataset





Example 5.2: Interpolations: top shows interpolation (sample 1-1) between bars 13 and 17 in sessiontune6 from the SESSION dataset using an RNN decoder, then in the middle, an interpolation (sample 1-4) between bars 1 and 4 in sessiontune14425 from SESSION dataset as well by an RNN decoder trained with scheduled sampling. At the bottom, an interpolation (sample 1-9) between bars 2 and 7 in sverige03 from the ESSEN dataset by a BALSTM decoder. RNN decoder trained with scheduled sampling and at the bottom a BALSTM decoder.

5.1.3 Long Runs (VAE and Context)

The same best-performing RNN and BALSTM models from the last section were now tested with the context conditioning included. Batch size has now been reduced to 128 due to the resource-heavy nature of the BALSTM where each original sequence is in reality internally divided into 96 sequences (one per pitch modelled), increasing the batch size with a factor of 96. A validation set of 10% of the training data is now used since there is a desire for the model to generalize given some contextual input. Preliminary results indicate that the models don't overfit using the current setups which is why dropout has been left out in this experiment too.

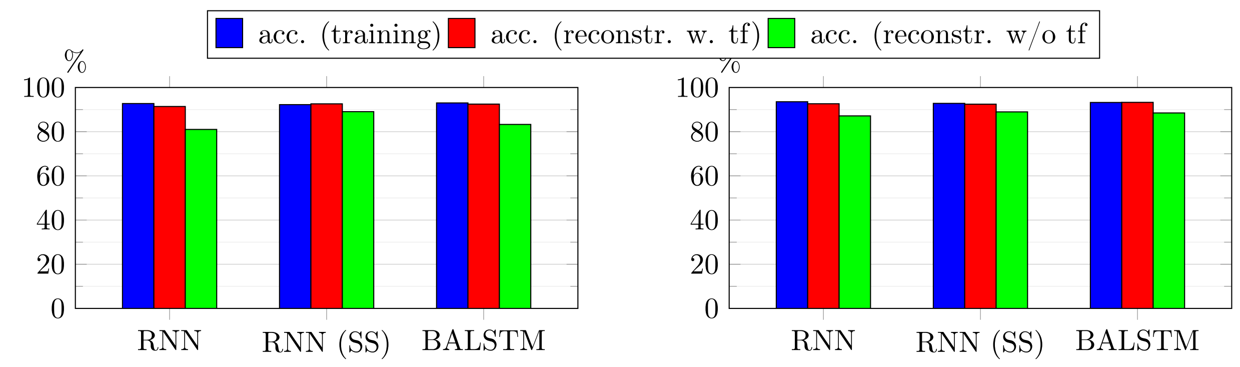
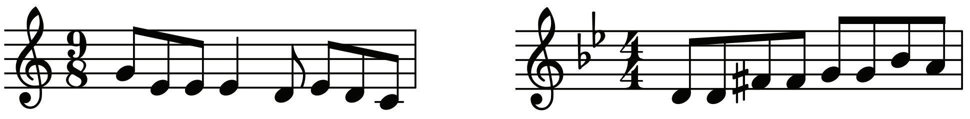
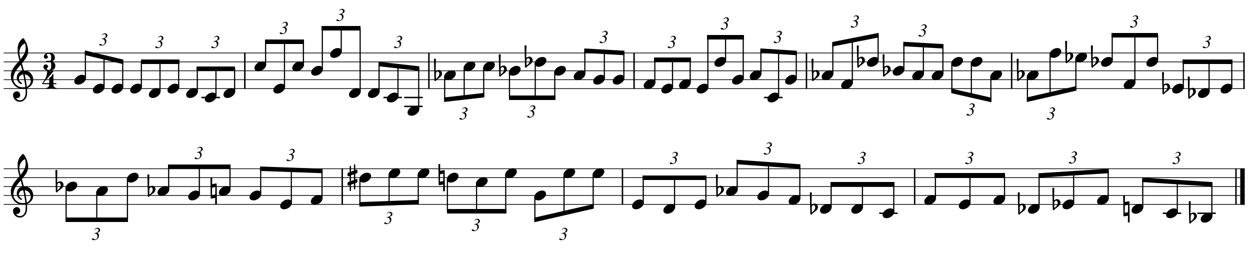


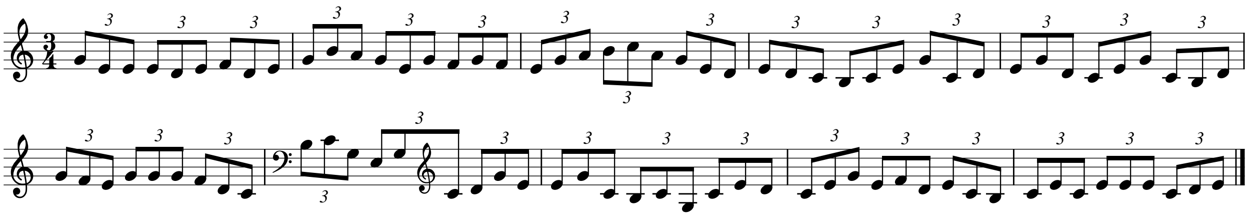
Figure 5.3: Comparison in reconstruction ability between RNN decoders with and without scheduled sampling and a BALSTM decoder. All models use a context that conditions the decoder VAE. The left plot is for the SESSION dataset and the right for the ESSEN dataset.

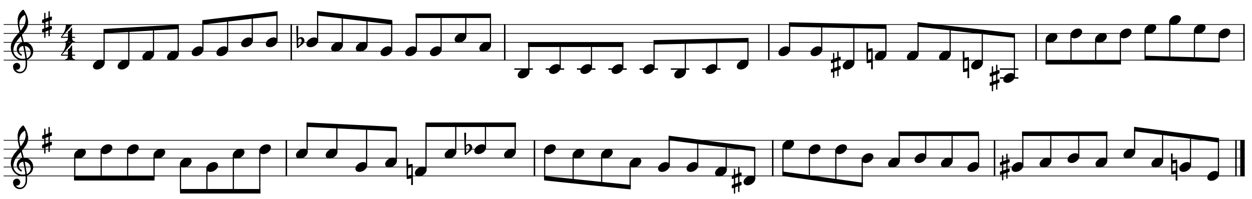
Next, 10-bar sequences were sampled from both the models with and without contextual input for comparison. For models without contextual input, this just amounts to repeated random sampling whereas for models with context input, the last sampled bar of music was fed as context for the next bar to predict, starting with the START token only in the context for the first bar. For the SESSION dataset, the first bar of sessiontune0 was used for seeding whereas in the case of the ESSEN dataset, the first bar of sverige06 was used. Both bars in original appearance is shown below.

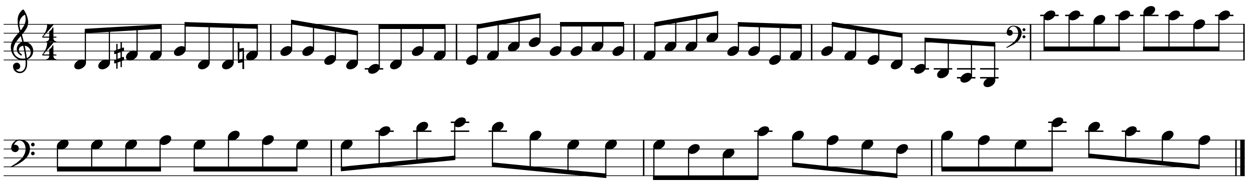


Example 5.3: The seeds used to yield a first-iteration latent vector. To the left, the opening bar of sessiontune0 from the SESSION dataset and to the right, the opening bar of sverige06 from the ESSEN dataset.









Example 5.4: 10-bar samples: top shows two 10-bar samples (samples 1-17 and 1-22) seeded with the first bar of sessiontune0 from the SESSION dataset. The first sample is done with a model that doesn't use context with an RNN decoder trained with scheduled sampling. Its counterpart is sampled in the same way but from a model trained with context and a regular RNN decoder, using a context with only the START token initially. The bottom two examples (samples 1-19 and 1-27) are generated in the same way but from the opening bar of sverige06 from the ESSEN dataset. The model without contextual conditioning here uses an RNN decoder whereas the sample with context is produced with a BALSTM decoder.

5.2 Experiment 2: Modelling All Properties

After having tested MahlerNet on pitch only, all modelled aspects were now included, modelling offset to previous event, event duration, event pitch and event instrument for every time step. A few changes were made to the initial basic model and inspired by the MusicVAE; the regularization by means of VAE loss was weakened to allow the model to yield better reconstructions at the price of a slightly less organized latent space. The parameters used were taken from MusicVAE in their initial experiments where both the free bits method as well as the β-VAE was used. The number of free bits allowed were 48 and the β parameter was initially set to 0.0 and growing slowly to reach its maximum value of 0.2 asymptotically as the number of global steps goes towards infinity. In reality, this yield an effective β of 0.08 at 50000 steps and 0.1148 at 100000 steps.

Decoder	**Encoder**	**VAE**	**Decoder**	**Learning rate**
RNN	2*512	256	2*512	0.001
BALSTM	2*256	256	2*256	0.001

Table 5.4: Basic setup for experiment 2.

For this experiment, three new datasets were used, namely three of the well-known baseline datasets often used.

Dataset	**samples**	**max seq. length**	**avg. seq. length**	**musical domain**
PIANOMIDI	47544	116	14.16	polyphonic
NOTTINGHAM	38339	21	8.12	homophonic
MUSEDATA	37006	158	26.25	polyphonic

Table 5.5: Information about the datasets used in experiment 2.

All of these datasets except MUSEDATA are modelled using a single or very few instruments. NOTTINGHAM contains mostly melody over a chordal background.

5.2.1 Short Runs (VAE Only)

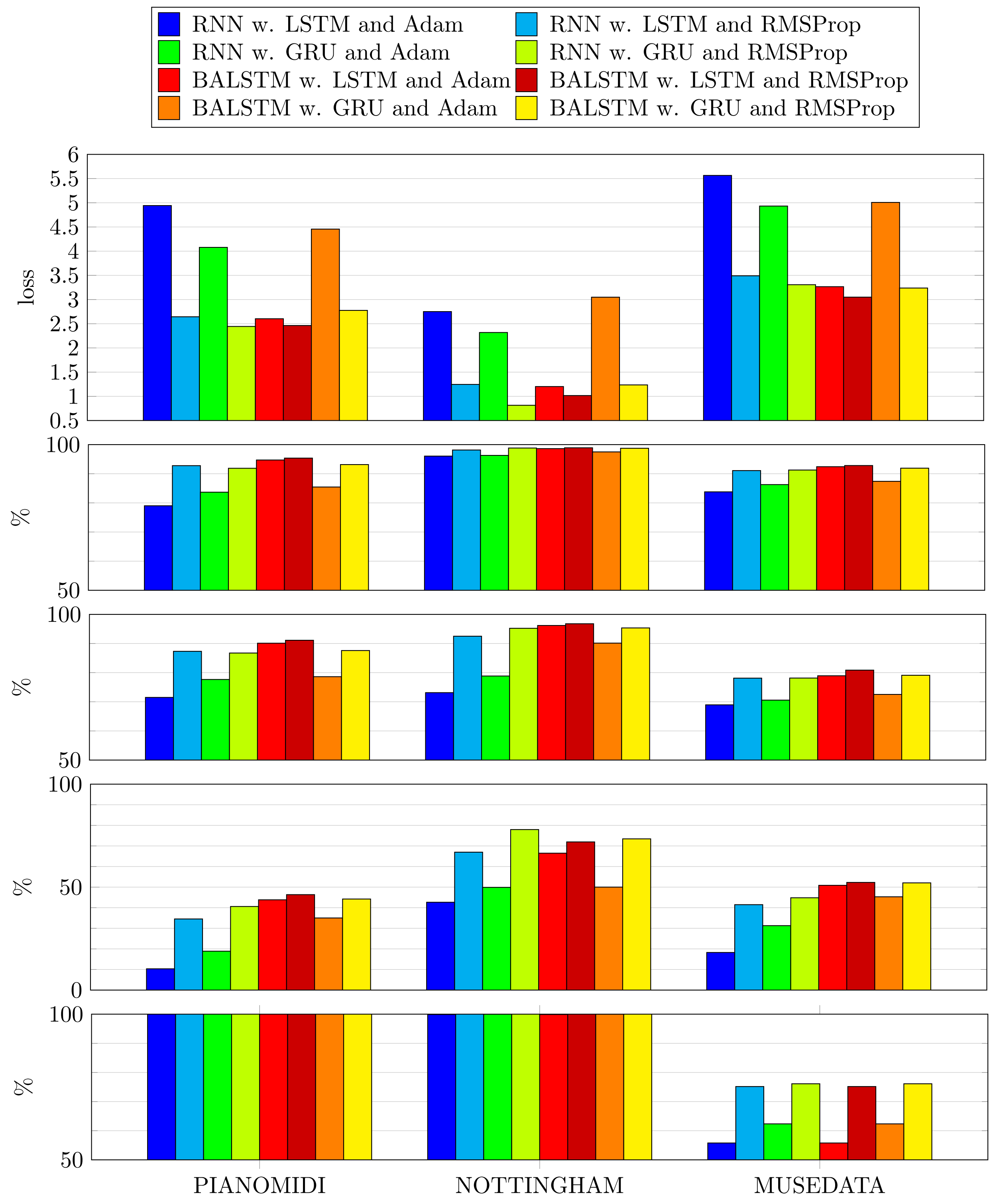


Figure 5.4: Comparison between RNN decoder and BALSTM decoder with different settings. Metrics are measured on full training set. Top shows loss and then teacher-forced reconstruction precision for offset, duration, pitch and instrument. Training lasted for 10 epochs for all RNN models. For the BALSTM corresponding value is 4 epochs. Values should not be compared between RNN and BALSTM due to differences in the training runs.

5.2.2 Improving the Results

The best setups from the short runs are now tested with four types of improvements: three conditionings during decoding and one reduction technique.

5.2.2.1 Conditioning on Metric Structure

The MUSEDATA and PIANOMIDI datasets are now used along with additional metric structure that mark every eighth note downbeats in an 8-class vector (corresponding to all the eighth notes in a 4/4 bar). The best setups from the previous section were used with the same settings apart from the extra beats vector.

The loss is hardly affected at all by the addition of beats but it is possible to see a slight improvement in prediction of offset and duration at times and unfortunately also a deterioration in pitch and instrument.

	RNN (GRU) w. RMSProp			*BALSTM (LSTM) w. RMSProp*			
	PIANOMIDI		*MUSEDATA*	*PIANOMIDI*	*MUSEDATA*		
	offset	*duration*	*duration*	*pitch*	*duration*	*pitch*	*instrument*
Default	91.88%	86.74%	78.15%	**46.35%**	80.87%	**52.28%**	**79.11%**
Beats	**93.01%**	**88.24%**	**79.73%**	44.73%	**81.74%**	51.57%	78.10%

Table 5.6: Effects of adding a vector with beats conditioning on metric structure.

5.2.2.2 Improving Pitch Predictions

Pitches occurring at the same time step are ordered from low to high (since it is more reasonable to condition higher pitches on lower than vice versa) by the preprocessor. MUSEDATA and PIANOMIDI are now tested with the best setups from the short runs but with the same step reduction feature activated. This feature masks out the logits of pitches lower than the previous one modelled, should the new event have an offset of zero, so that the pitches are started simultaneously. The other way, to improve pitch by conditioning on the currently sounding pitches (if any) is also tried out.

	RNN (GRU) w. RMSProp		*BALSTM (LSTM) w. RMSProp*	
	PIANOMIDI	*MUSEDATA*	*PIANOMIDI*	*MUSEDATA*
Default	40.59%	44.83%	**46.35%**	52.28%
Same step reduction	40.08%	41.27%	45.13%	52.84%
Active pitches	**41.22%**	**45.76%**	45.93%	**53.06%**

Table 5.7: The effects of conditioning on currently sounding pitches as well as reducing the available pitches whenever offset from the previous event is zero.

5.2.2.3 Conditioning on Active Instrument

This section uses a conditioning on active instruments instead of active pitches, as was the feature to evaluate in the last subsection. Also, only MUSEDATA has been evaluated since PIANOMIDI only contains piano music.

	RNN (GRU) w. RMSProp	*BALSTM (LSTM) w. RMSProp*
Default	76.11%	79.11%
Active instruments	**77.45%**	**80.37%**

Table 5.8: Teacher-forced reconstruction accuracy for instrument when modelling all properties and with and without active instrument conditioning.

5.2.2.4 The New Default

The techniques previously presented that had a neutral or positive impact on the outcome was incorporated in the default for the rest of the experiments. This means that all the conditionings were kept whereas the reduction technique on pitch was removed. Below is presented the impact of employing all three conditionings at the same time.

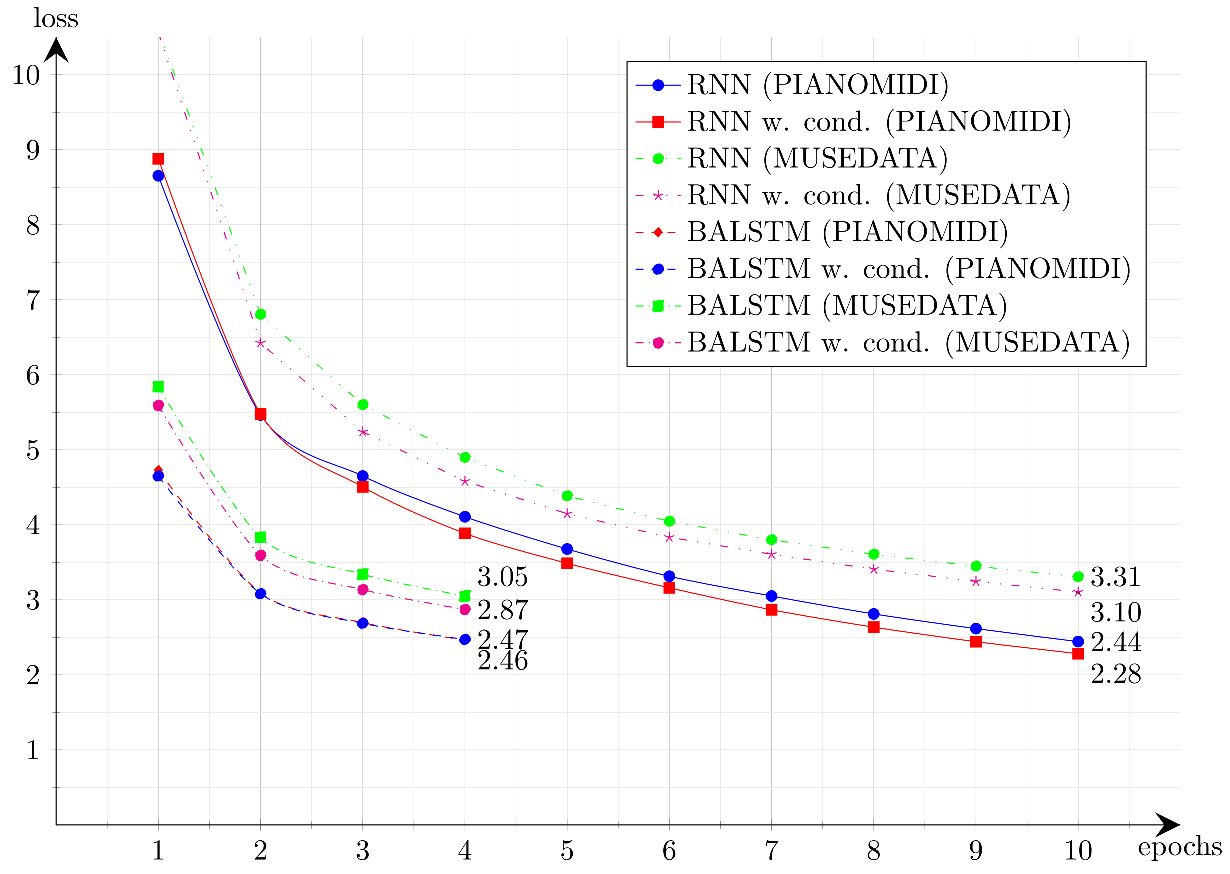


Figure 5.5: All setups except for one with the BALSTM benefits from decoding with additional conditionings on metric structure, active pitches and active instruments.

	RNN (GRU) w. RMSProp		*BALSTM (LSTM) w. RMSProp*	
	PIANOMIDI	*MUSEDATA*	*PIANOMIDI*	*MUSEDATA*
Offset	91.88% / **93.82%**	91.28% / **93.18%**	**95.38%** / 95.20%	92.82% / **93.98%**
Duration	86.74% / **88.45%**	78.15% / **79.96%**	91.12% / **91.58%**	80.87% / **82.56%**
Pitch	40.59% / **43.00%**	44.83% / **46.24%**	**46.35%** / 45.87%	52.28% / **53.84%**
Instrument	**100.00%**	76.11% / **77.56%**	**100.00%**	79.11% / **80.60%**

Table 5.9: Results from supplying three conditionings during decoding.

5.2.3 Long Runs (VAE Only)

Longer runs with the best setups with conditionings were now undertaken. Since convergence was slow, batch normalization was added before the activation function of all fully-connected layers in the network, except in those belonging to the VAE, including the output layers. In conjunction to this, the activation function was switched to leaky ReLU instead of tanh and both the BALSTM and RNN were assigned LSTM cells that now outperformed the GRUs. Recurrent batch normalization was attempted as well but yielded less improvement in performance.

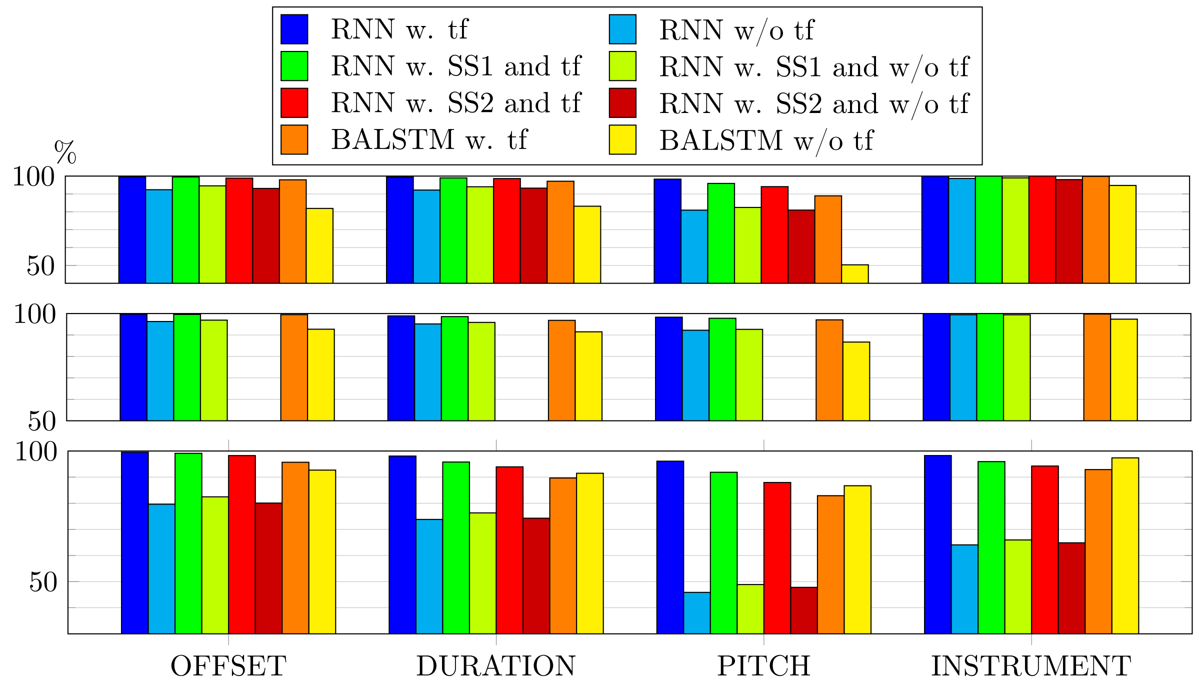


Figure 5.6: Pairwise comparisons between teacher-forced and non-teacher-forced reconstruction of the entire datasets. From top to bottom, reconstruction percentages with respect to modelled properties for PIANOMIDI, NOTTINGHAM and MUSEDATA datasets. PIANOMIDI and MUSEDATA were tested with two different settings for scheduled sampling (linear schedule with 0.00001 and 0.000025 decay) whereas NOTTINGHAM only used one setting (inverse sigmoid schedule with rates 1000 and 1500). RNN training lasted for 50 epochs for PIANOMIDI and MUSEDATA and 30 epochs for NOTTINGHAM. With the BALSTM, training lasted for 40 epochs for MUSEDATA, 30 epochs for PIANOMIDI and 24 epochs for NOTTINGHAM.

5.2.3.1 BALSTM Transpositional Invariance

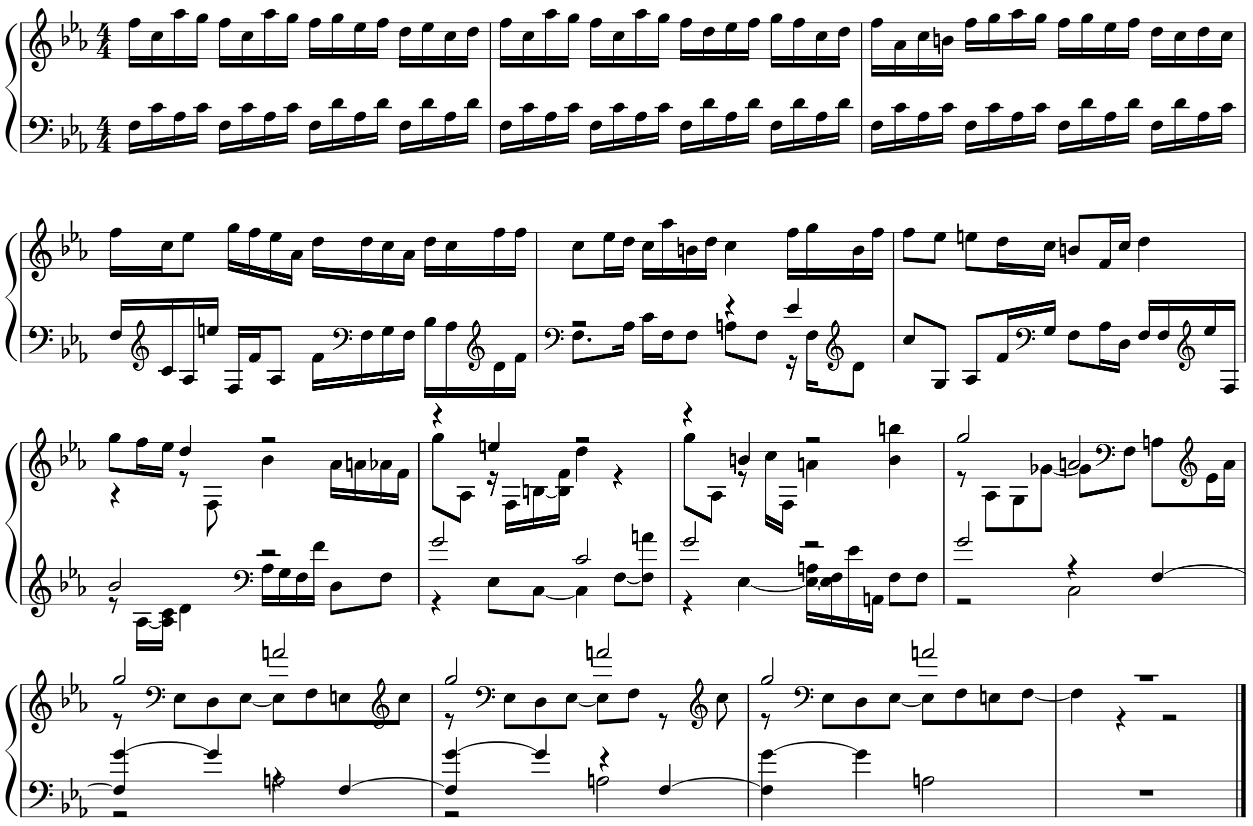
The PIANOMIDI dataset was transposed four steps down and both the BALSTM and RNN decoder from last section was used to reconstruct the full set of 47544 samples. Given the unanimous results with respect to the BALSTM, it was dropped in the parts left of the experiments.

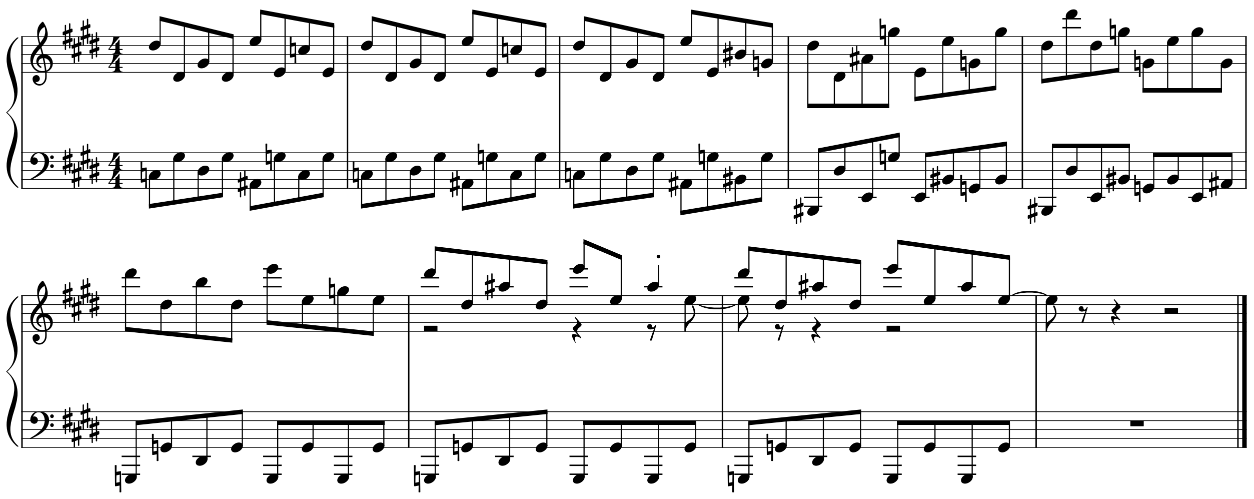
decoder	w. tf	w/o tf
RNN	59.54%	37.81%
BALSTM	73.83%	31.04%

Table 5.10: Reconstruction accuracy on a transposed version of the PIANOMIDI dataset with and without teacher forcing.

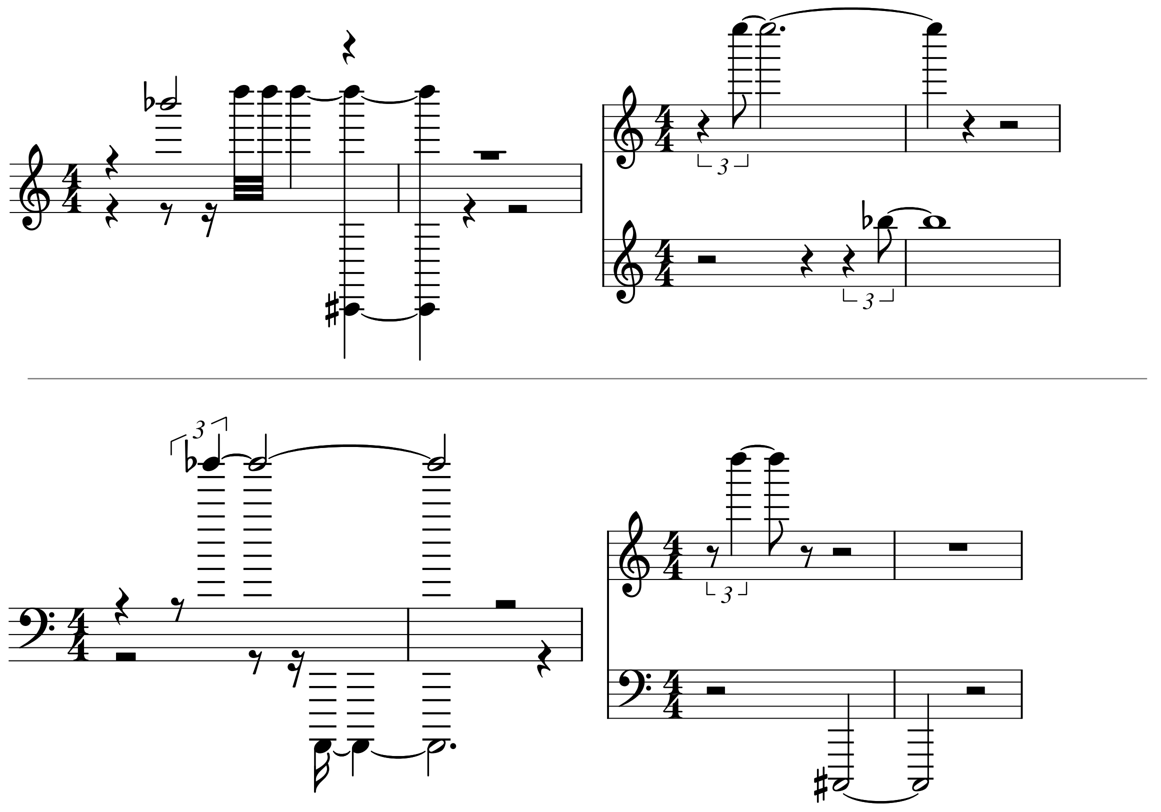
5.2.3.2 VAE Latent Space

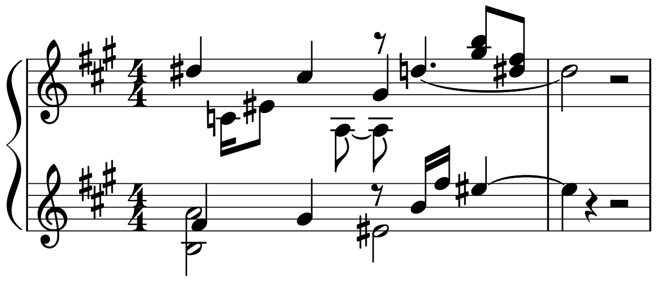
Interpolations were done with models trained on the PIANOMIDI and MUSEDATA datasets. No samples from MUSEDATA are shown due to their size since most often, several instruments are included in the generated music. They can however be heard at the MahlerNet website at `https://www.mahlernet.se`.




Example 5.5: Interpolations: top shows interpolation (sample 2-1) between bars 136 and 203 in mz_ 333_ 1 from the PIANOMIDI dataset using an RNN decoder and then below, an interpolation (sample 2-6) between bars 105 and 110 in pathetique_ 1 from the PIANOMIDI dataset as well but this time by an RNN decoder trained with scheduled sampling.

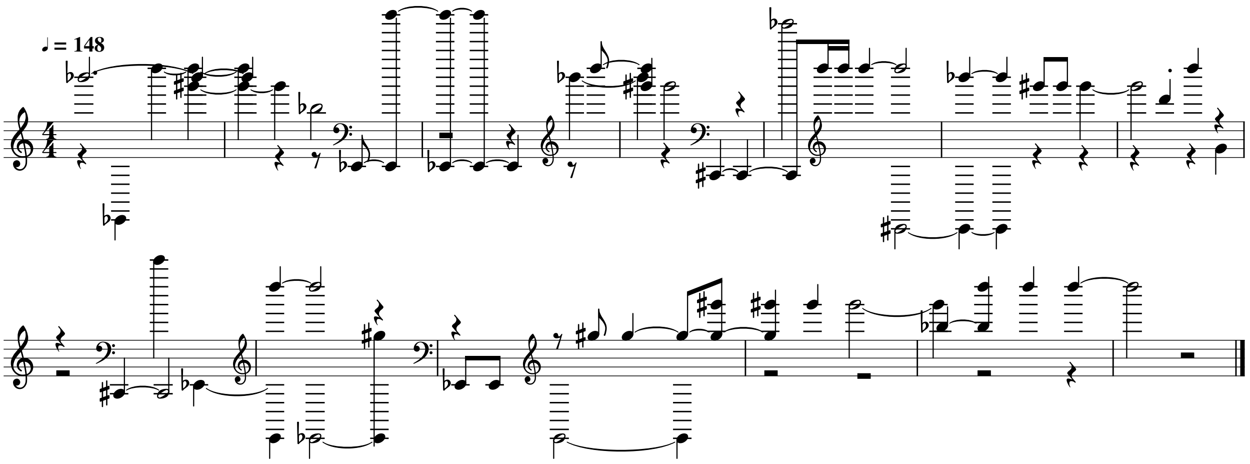
Next, as in the previous experiment, random samples were drawn from the models.

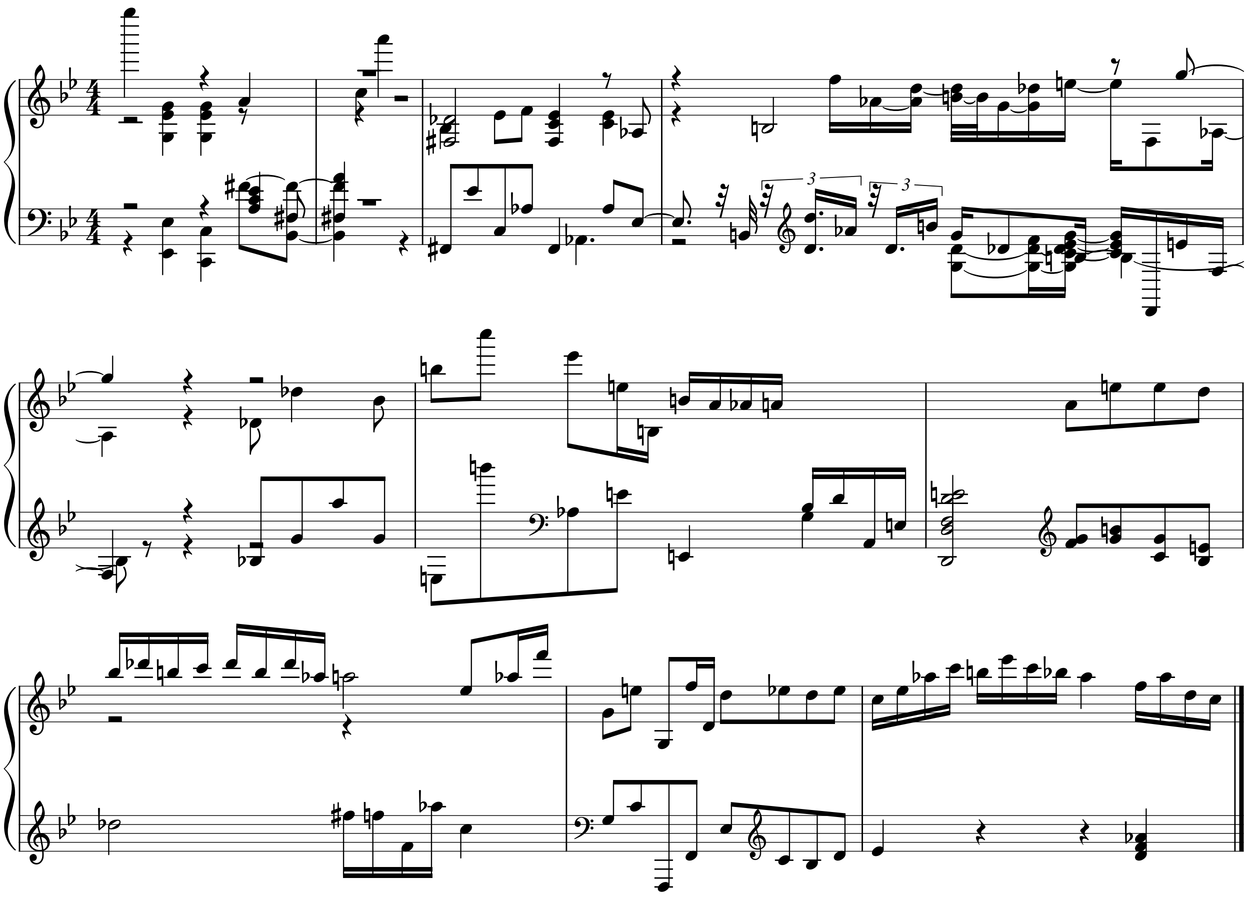




Example 5.6: Sampled bars from MahlerNet with an RNN decoder on top (samples 2-3 and 2-13), then an RNN decoder trained with scheduled sampling (samples 2-7 and 2-15) and at the bottom row, sample from a BALSTM decoder (sample 2-10). The left column is from models trained on PIANOMIDI and the right from models trained on MUSEDATA. The BALSTM sample is only one to the number and from a model trained on PIANOMIDI since the sample from the model trained on MUSEDATA was too big to fit here.

As a follow-up on the sampled bars, a 10-bar sample (and since there is no context, this means ten sequential but uncorrelated samples) was drawn from the model with an RNN decoder trained on PIANOMIDI and compared to a 10-bar sample drawn from a new model trained on PIANOMIDI with the same settings as the former, but with the optional use of β-VAE and the free bits technique turned off. This means that the latter model employs the default VAE loss as was done in experiment 1.

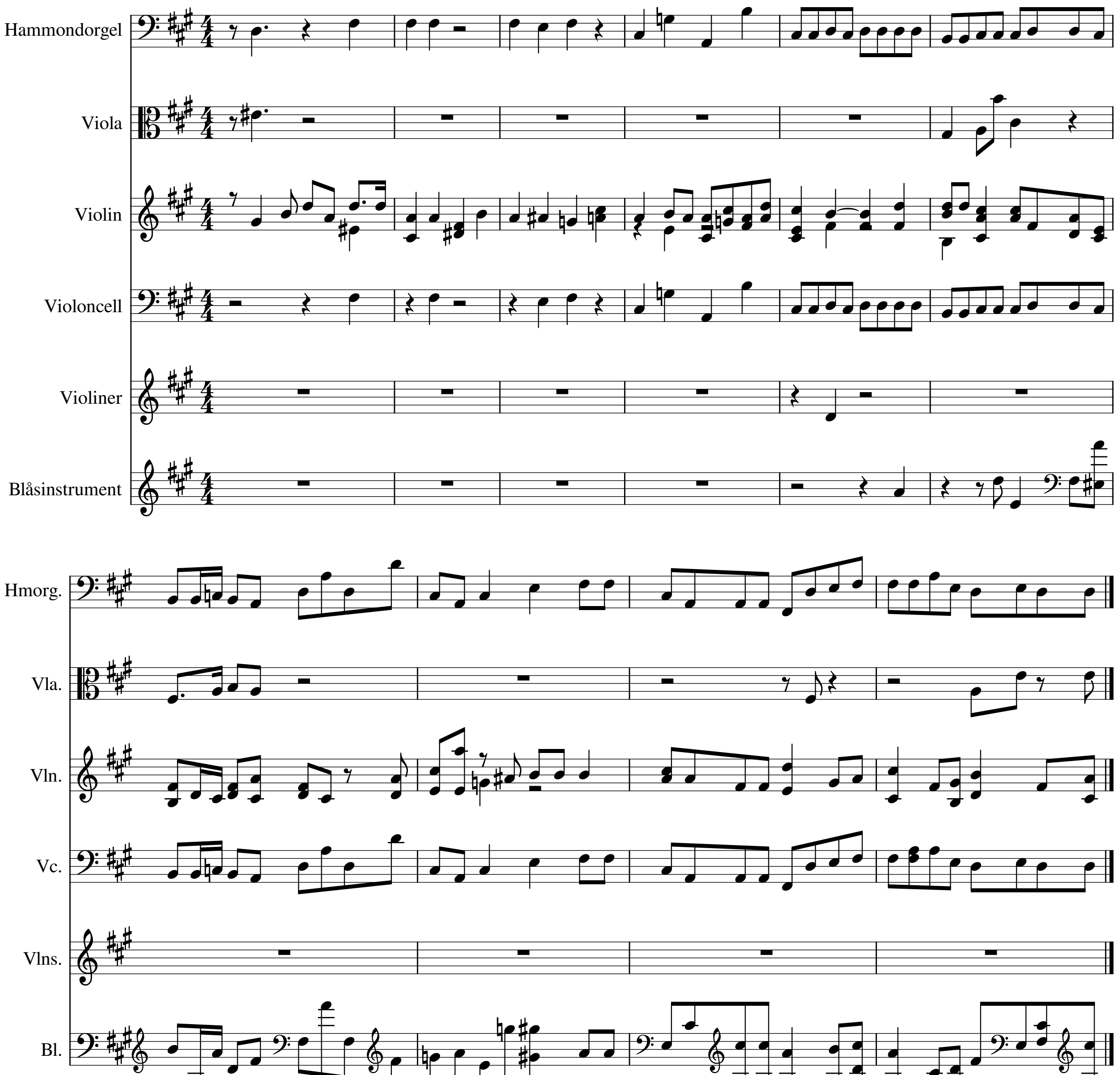




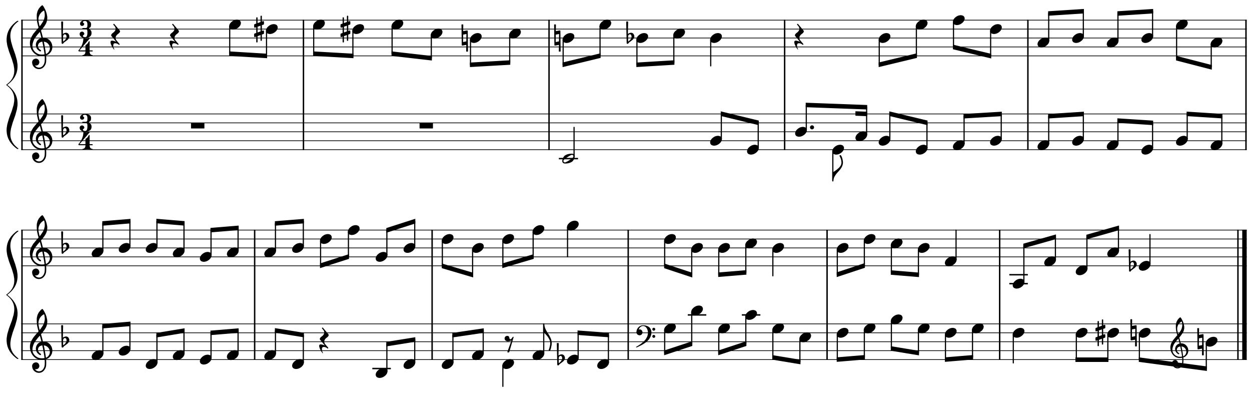
Example 5.7: At the top, a 10-bar sample (sample 2-4) drawn from a model with RNN decoder and trained on PIANOMIDI with 48 free bits and β annealing in the VAE loss. At the bottom (sample 2-11), a model with the same settings but with free bits set to 0 and β-annealing entirely turned off. Since no context conditioning occurs, there is no expectation on the bars to be correlated and they are sampled one and one and concatenated afterwards.

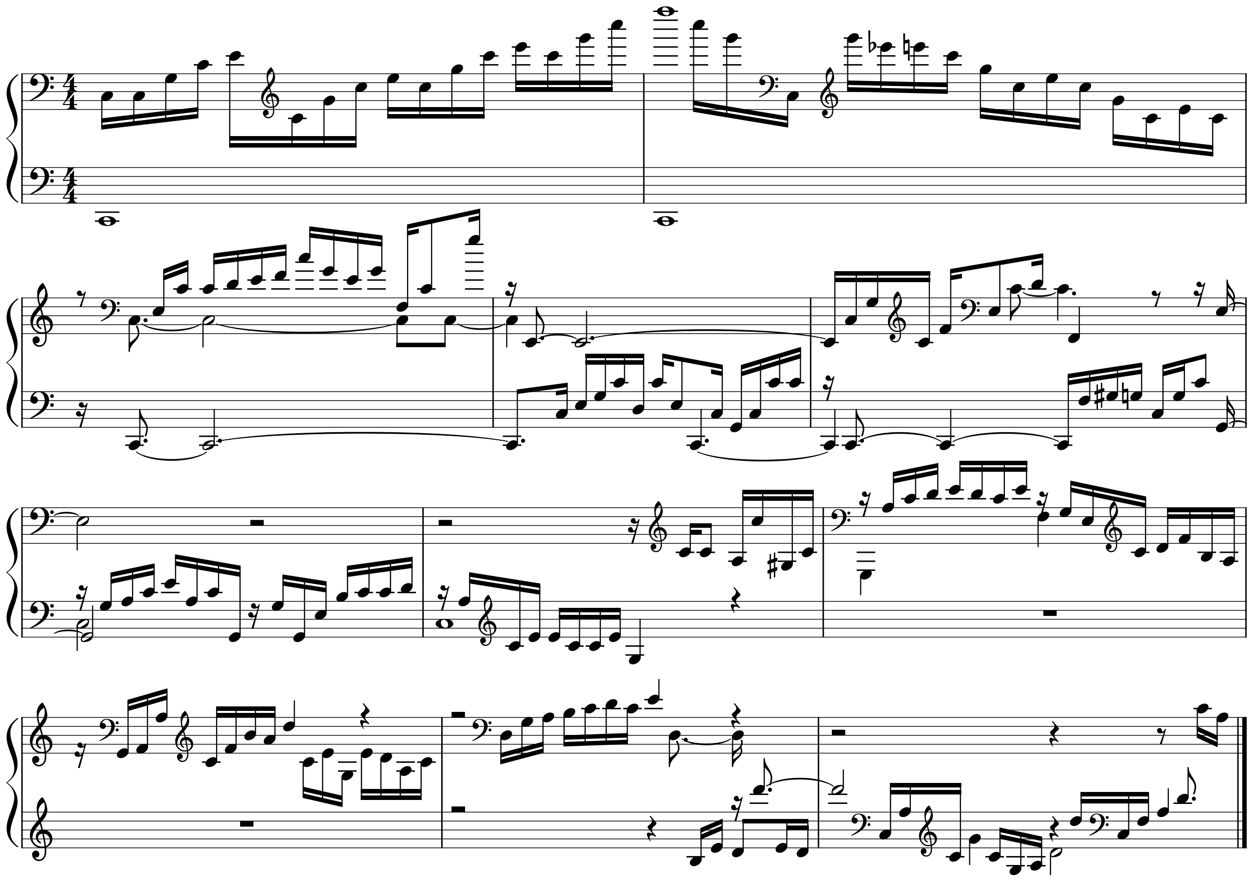
5.2.4 Long Runs (VAE and Context)

Long runs using context were now performed. Due to the BALSTM taking longer to train and performing bad, it was excluded from further experiments. Furthermore, the β-VAE and free bits technique were removed in the VAE loss and dropout of 35% was introduced in every layer (resulting in the processing order weight multiplication $\Rightarrow$ batch normalization $\Rightarrow$ activation function $\Rightarrow$ dropout) of the network except for in the input and output layers. Models are only trained on PIANOMIDI and MUSEDATA and only two versions exist for each: with and without scheduled sampling. Models trained until the loss of the validation set started to show tendencies to increase, after which the saved model from the last epoch before this was used for inference. This resulted in models training on the PIANOMIDI dataset for 15 epochs for the regular model and 22 epochs for the model trained with scheduled sampling. Corresponding training lengths for MUSEDATA was 21 and 25 epochs.

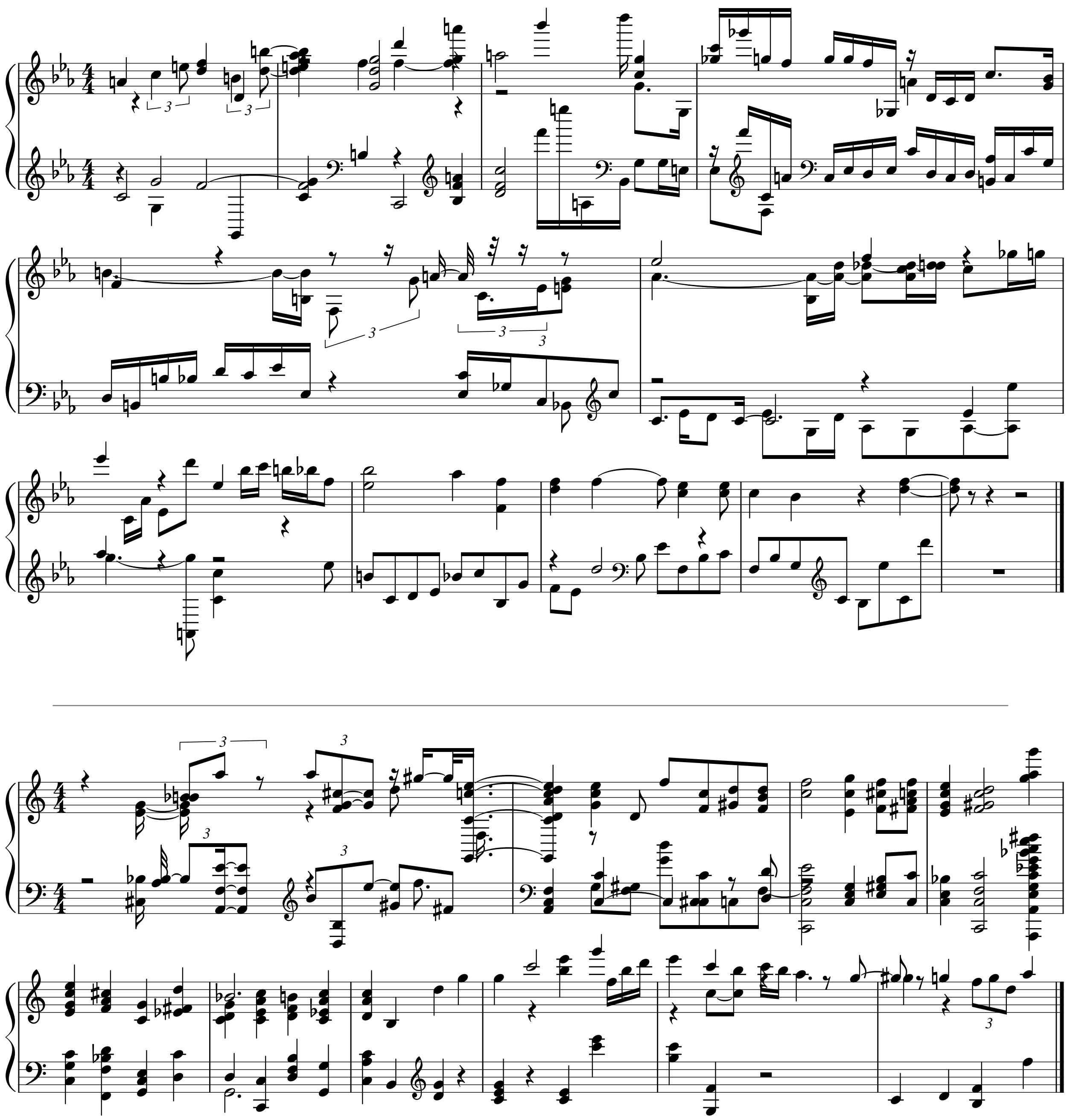


Example 5.8: 10-bar random sample (sample 2-56) drawn from a model conditioned on context with RNN decoder and trained on MUSEDATA with standard VAE loss. The sample was randomly generated with an initial context with a START token only and sampled latent vectors all along, using softmax temperatures 0.7, 0.7, 0.6 and 0.7 for offset, duration, pitch and instrument.





Example 5.9: 10-bar samples (samples 2-18 and 2-32) drawn from models conditioned on context with RNN decoder and trained on PIANOMIDI with standard VAE loss. The model producing the bottom sample was also trained with scheduled sampling. For top sample, the model was seeded with bars 1 and 2 for context and initial input from MIDI file elise, using softmax temperatures 0.9, 0.9, 0.7 and 0.5 for offset, duration, pitch and instrument. The model producing the bottom sample was seeded with bars 1 and 2 from chpn_ op10_ e01 with all softmax temperatures set to 1.0.



Example 5.10: 10-bar samples (samples 2-26 and 2-39) drawn models conditioned on context with RNN decoder and trained on PIANOMIDI with standard VAE loss. The model producing the bottom sample was also trained with scheduled sampling.

Both seeded and random samples were taken. A few are presented above but the reader is urged to listen to output at the MahlerNet website at `http://www.mahlernet.se` instead, due to lack of space in this writing.

5.3 Experiment 3: Modelling Gustav Mahler

In the final experiment, essentially the same settings from the last portion of experiment 2 was used. Only two long runs were made: one without context and one with.

Decoder	Encoder	VAE	Decoder	Learning rate	Dropout
RNN	2*512	256	2*512	0.001	35%

Table 5.11: Basic setup for experiment 3.

The MAHLER dataset was collected from three Internet sources for the purpose of this work and contains all of Mahler's symphonies except for a few movements that was not found.

Dataset	samples	max seq. length	avg. seq. length	musical domain
MAHLER	17450	303	29.01	polyphonic

Table 5.12: Information about the dataset used in experiment 3.

5.3.1 Long Runs (VAE Only)

As in experiment 2, the models were trained for 50 epochs using one model with an RNN decoder and one model with an RNN decoder trained with scheduled sampling.

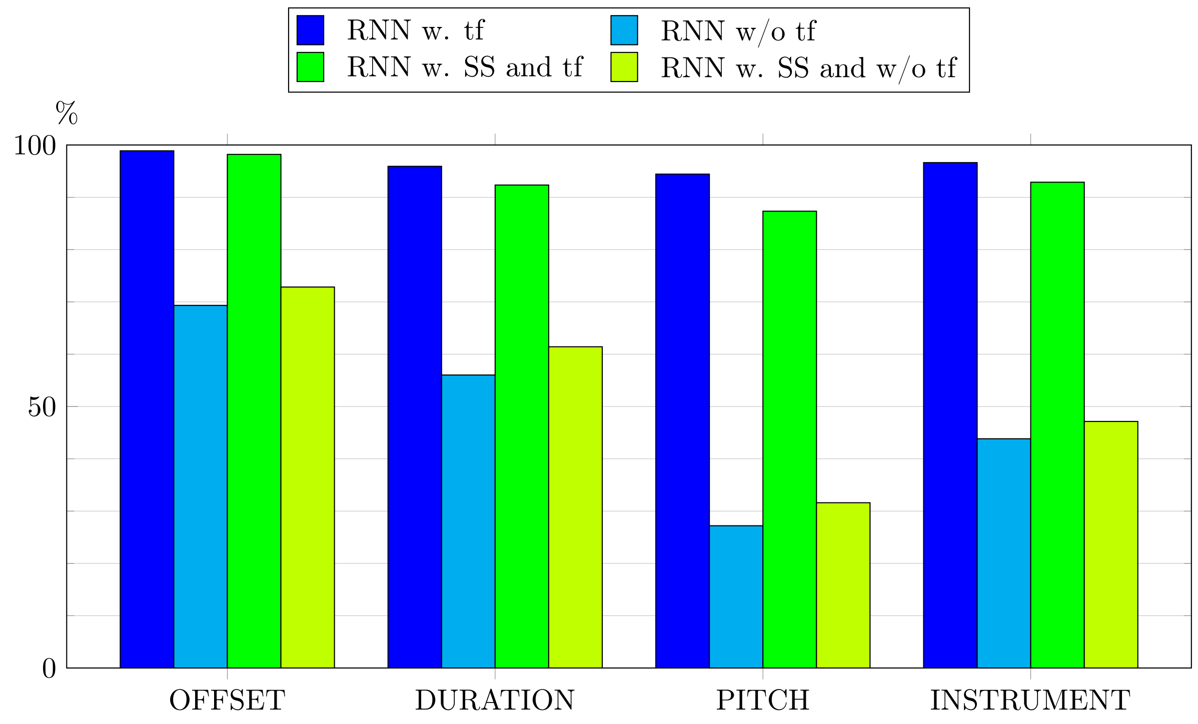
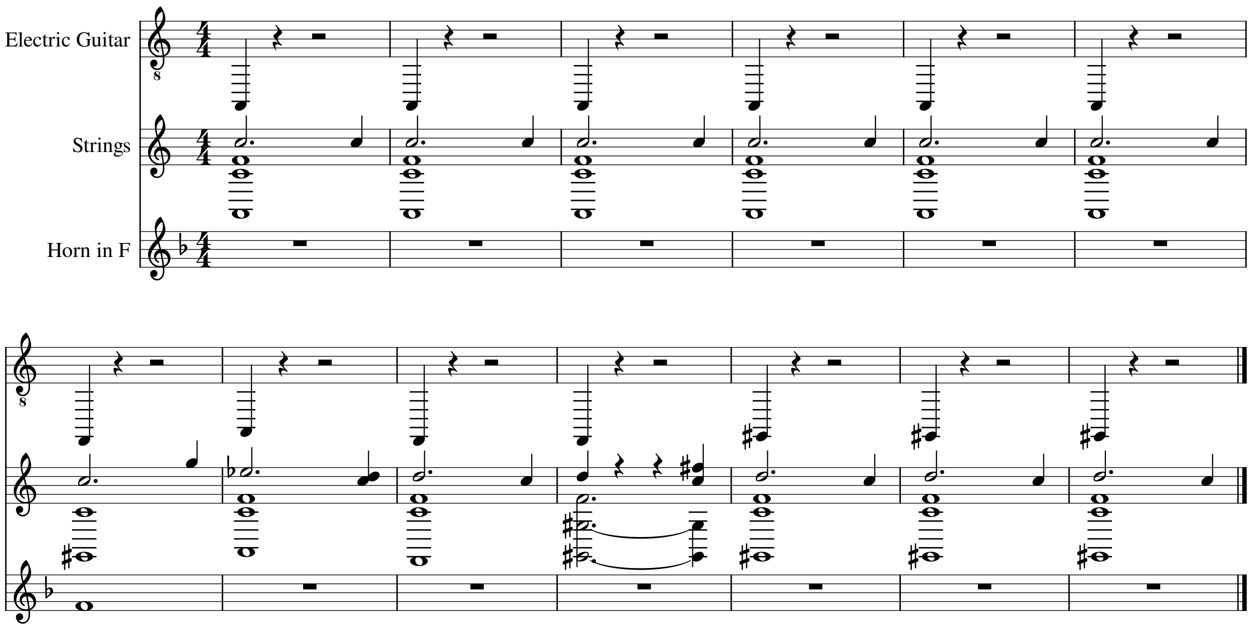


Figure 5.7: Pairwise comparisons between teacher-forced and non-teacher-forced reconstruction of the entire MAHLER set. The model trained with scheduled sampling had a linear schedule with 0.00001 decay. Training lasted for 50 epochs.

5.3.1.1 VAE Latent Space

As usual, interpolations were done with input bars from the MAHLER dataset. There is not room to show results here but they can be heard at the MahlerNet website `http://www.mahlernet.se`. In general, interpolations were not very smooth for bars with a lot of events but bars with less events were successful.

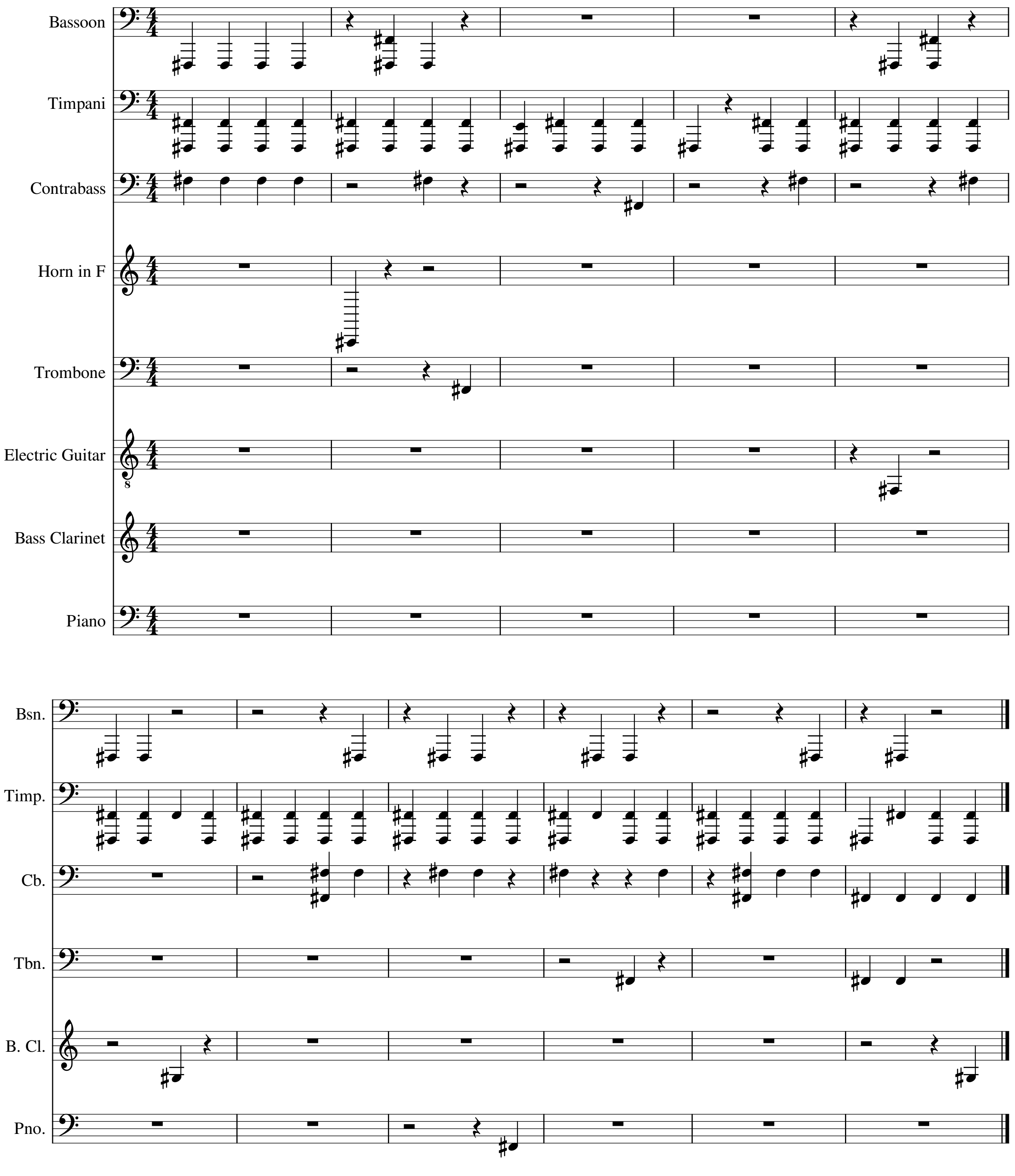


Example 5.11: 10-step interpolation (sample 3-1) (resulting in 13 bars including copied in end-points) between bars 9 and 10 in Mahlsy54 from the MAHLER dataset. Interpolation was done by a model using an RNN decoder and trained for 50 epochs with batch normalization and without scheduled sampling. The number of events in the bars are few and the interpolation is smooth between endpoints and endpoints are correctly reconstructed.

5.3.2 Long Runs (VAE and Context)

In the long runs, the same approach as for the long runs with conditioning in experiment 2 was used. The model trained without scheduled sampling was optimal at 31 epochs whereas using a linear scheme for schedule sampling with rate 0.000010, the optimal model had trained for 33 epochs. Both batch normalization and a dropout of 35% were used for both models.

Listening samples en masse exist on the MahlerNet website `http://www.mahlernet.se`. Due to the samples being large, they do not fit in this report. All samples were generated with softmax temperatures between 0.5 and 0.8 for all properties.



Example 5.12: 10-bar random sample (sample 3-45) from a model using an RNN decoder and trained with scheduled sampling using a linear schedule with a 0.00001 linear coefficient. The model was seeded with bar 553 as initial context and 554 as initial latent sample from Mahler61. The sample is not very interesting but since, except for the first bar, the content is sampled from the latent space, it shows that the context manages to create a long-term structure in terms of instruments used as well as offsets, durations and pitch content.

6 DISCUSSION

6.1 Implementation

6.1.1 MIDI

Someone who hasn't worked in detail with the MIDI format might believe that it lies closer to written than sounding music and that a conversion from MIDI to a data representation of the corresponding symbolic music is straightforward. It turns out that this is, by far, not the case. MIDI is in fact a very rich data representation that can express music in much more detail than can written music, and thus it is almost as if it was developed with sounding music in mind rather than written. For example, dynamics in MIDI is expressed as a number between 0 and 128 which is richer than the classical terms most often used for written music. Furthermore, the default MIDI setting uses 96 ticks for a quarter beat which effectively allows for a quarter to be divided into 96 parts whereas it is very rare to see written music with anything smaller than 32nd notes, which are also rare. MIDI thus lends itself to capturing the details of sounding music and in general, it seems like the format has been used more with this aspect in mind rather than as an alternative format for written music.

Another important aspect is how MIDI files are created and with what intention. Given a sequencer or music writing software capable of generating MIDI files, there is no silent rule saying that generated MIDI must correspond exactly to the written music and so, the software could potentially create the file with the ambition that it should sound good and therefore use heuristics to add artifacts to the output MIDI so that it sounds more like music played by humans. When MIDI files are recorded by humans on MIDI instruments, the generated music is per definition a capture of sounding music rather than written unless some heavy quantization has been used. And what stops the human performer from adding things that make the music sound more like it would have sounded if it would have been played by a band or an orchestra, at the cost of bringing it even further away from its written counterpart?

It very soon becomes obvious when processing any MIDI file with the slightest complexity that a whole arsenal of tricks and cheats are used to make the listening experience more enjoyable, sacrificing the symbolic precision of the very same piece. Examples of tricks are doubling the length of all notes in a portion of the music to illustrate a halved tempo (instead of actually changing the tempo), spelling out tremolos and trills in a way that breaks how the same passage is notated in the score and doubling voices in more MIDI instruments than what is done in the original manuscript. Any project using MIDI to create a representation of the corresponding written music (MIDI has also been used to model sounding music ***(Eppe et al., 2018)***) must then make use of a powerful preprocessor that creates a data representation of the music that is meaningful, correct and with enough of the original retained and as little as possible of the added artifacts kept. The MIDI format is thus not ideal for modelling symbolic written music and it results in an unnaturally large responsibility placed on the MIDI preprocessor.

6.1.2 Architecture

The conditions listed in section 1.2 were all fulfilled with the following comments:

- Condition C.2: 23 instruments were used in this work since some concrete number had to be decided upon. However, this number is arbitrary and the design is by no means restricted to some special number, hence condition C.2 is fulfilled to the desired extent.
- Condition C.3: In this work, the context conditioning and a latent vector calculated from an input bar (as opposed to randomized) act as the kind of the seeding and conditioning as condition C.3 refers to. Thus, this condition is fulfilled but more importantly, MahlerNet has an adaptable design which can easily be extended to allow further arbitrary conditioning, for example expressing musical style or time signature, if wanted.
- Condition C.6: As mentioned in section 4, even though minimal assumptions have been placed on the input music, there are some assumptions since we are, in fact, targeting Western music. This condition is thus fulfilled.

The architecture was repeatedly developed starting out with neither scheduled sampling nor batch normalization or dropout implemented. Early on all durations were modelled with a distributed representation where only 11 out of the basic set of durations were used as features. No matter how intuitively appealing and efficient this may seem, switching to a one-hot representation with 60 classes and softmax instead of sigmoid as output function immediately lead to an improvement in performance. One reason for this might be that when durations are modelled in a distributed way, the zero duration is modelled only as the lack of all other durations whereas when using a local representation, this duration is actively represented by a class. Perhaps this has a positive impact on performance and with this duration explicitly modelled in the distributed representation as well, this representation would work better. Nonetheless, this would imply other oddities in the representation that would have to be taken care of (for example, how to interpret an active zero duration when other duration classes are active as well).

From a musical point of view, the order of conditioning among the modelled properties may seem somewhat arbitrary, but some rule of guidance can be found. First of all, predicting instrument should be conditioned on pitch since the choice of instrument is highly dependent on in which range the pitch to play lies. Pitch, in turn, depends on offset to previous event since the closer to the previous event, the less likely it is idiomatically that the new pitch is far away from the last. When it comes to pitch and duration, conditioning can go both ways; one might argue that given a long duration, it is more likely that the pitch is in the key of the piece and vice versa and in the early versions of MahlerNet, pitch was modelled conditioned only on offset after which duration and instrument was predicted (what instrument to pick depends also on the duration since some instruments have a harder time playing short notes than others and vice versa). One way to go would be to model pitch and duration separately, but given the arguments above, the question is only about in *what way* the conditioning should go, not as to *if* there is a conditioning in the first place. Therefore, any conditioning is better than none. Nevertheless, keeping in mind that the order of conditioning is offset to duration to pitch in BachProp, the modelling order applied in MahlerNet was switched in later versions to use the same order with instrument predicted last. One other advantage of this was also that an empty time step, used to progress time, could now be signalled by using the zero duration, which would otherwise be redundant. With the zero step as a duration feature, both offset and duration could be modelled more similarly (instead of taking special care of the zero duration for the duration property) and there was no need for an extra "no pitch" class that would break the BALSTM pitch convolutions in terms of note vicinity.

Some of the arguments above do not necessarily hold true for the data representation chosen. For example, a short offset makes a short leap more likely only if the previous pitch was in the same melody line as the next, something that is not necessarily true in a representation where all vertical events are modelled sequentially. Thus, the last event may be from another melody line and then, a short offset does not mean anything.

6.2 Experiments

During training, an initial indication sought after was a model's capability to overfit. When not conditioning on context, the notion of overfitting becomes somewhat obscure and therefore, no specific attention was paid to overfitting; the models were trained for as long as possible to be able to represent the music they were trained on to the highest degree possible. After training, reconstructing the training set with the same accuracy as was measured during training, but now with the inference pipeline as opposed to the training pipeline using manual teacher forcing, was also important and indicative of whether the implementation was correct. This measure has therefore been used several times to illustrate this fact.

6.2.1 Experiment 1

Pitch was modelled using large datasets with sequences of short length. It is possible to see a clear indication of which model is worth to continue to explore in the short runs and for the RNN decoder, this turned out to be a model using GRU units with the RMSProp optimizer for both datasets. For the BALSTM, the GRU units were also winning but with the Adam optimizer for the SESSION dataset and RMSProp for the ESSEN dataset. For the BALSTM, it is important to remember that since each original sequence is divided into 96 sequences for single pitches, the underlying modelling RNN trains on much more data than does the regular RNN decoder.

When testing the winning models on long runs, it is obvious that the inference pipeline works as intended since the teacher-forced reconstruction percentages are about the same as the training accuracy for all models. However, it is also clear from this run that the use of scheduled sampling results in a higher full reconstruction accuracy, especially for the SESSION dataset. Here an inverse sigmoid rate of 1000 was used which is a sampling schedule that drastically increases the chance of sampling from the model predictions instead of the targets from the 5000th step and on.

The preliminary tests of the BALSTM transpositional invariance property revealed that the RNN decoder performed better with a transposed dataset, both with and without teacher forcing. Reasons for this might be that the context in which the BALSTM-like component is deployed differs significantly from where it is originally deployed and because of this, the results might suffer. First of all, the original BALSTM processes all events from the previous moment in time simultaneously which MahlerNet doesn't. A convolution over several pitches from the previous moment in time (which in the original BALSTM is the same as the previous time step whereas in MahlerNet, the previous moment in time can be spread out over multiple time steps in the model) is probably more indicative of whether a specific note should be turned on or not than a convolution over a range where only one pitch is turned on, which is the unavoidable result from modelling polyphony sequentially. This means that every RNN instance modelling a pitch has very little information about pitch context which might result in *some* kind of transpositional pitch invariance but a very suggestive one in an otherwise weak model. Also different from the original BALSTM is that the output of each BALSTM instance is concatenated and run through a softmax layer which is different from the binary output, indicating pitch activation or not for a given instance, in the original BALSTM. All in all, these changes might have a negative impact on the BALSTM, even though there was a hope, at this point, that it would perform better when modelling all properties. Especially so because it would then have access to the active pitches input which, similarly to the original BALSTM contextual setting, holds multiple simultaneous pitches that are turned on.

In this experiment, the standard VAE loss was used which is sometimes criticized for putting too much emphasis on a smooth and ordered latent space at the price of reconstruction quality. As expetced, the pitch experiment shows that the latent space of the VAE is smooth and ordered, even though some small deviations exist. The endpoints in the interpolations can be analyzed with respect to key and motives and often, there is a gradual transition from the first key and direction of motives to the second one and the endpoints are reconstructed with high accuracy. The BALSTM decoder seems to deliver more "imaginative" solutions, less idiomatic than the RNN decoders, both with and without scheduled sampling. Random samples from the latent space show that it is populated and forms a multidimensional surface where all sampled points correspond to music similar to encountered samples in the training data.

The addition of context conditioning in experiment 1 does not result in any wonders but it is clear that more convincing music is created from models conditioned on context compared to random concatenated samples which indicates that the context conditioning has a positive impact. Most of the time, the music stays somewhat in key with context conditioning which is obviously not the case without it. Melody lines are smoother with less large leaps between bars and in some places, motives are regurgitated in neighbouring bars.

6.2.2 Experiment 2

In experiment 2, already after the short runs, it is possible to see more diverse results than in experiment 1. First of all, the datasets used a fairly small but with long sequences and the performance of the models vary a lot over the datasets. The MUSEDATA dataset seems to be the hardest to model after which follows PIANOMIDI. Common for these two sets is that they contain longer sequences than other datasets from both experiments. Nevertheless, obvious indications regarding which models perform best can be spotted and yet again, RMSProp as an optimizer proves to be best. Apart from that, the RNN decoder uses GRU units whereas the BALSTM decoder uses LSTM units.

Before proceeding, some additional features of MahlerNet were tried out to decide on which of these to use for the remaining experiments. As expected, most of the features turned out to have an almost unanimous positive impact on performance, with one exception. The reduction of pitches to choose from as a result of zero offsets resulted in worse performance even though removing some of the possibly wrong predictions, in theory, shouldn't have anything but either a positive or no impact, yet here it has a negative impact. The reason for this might be that the weights connected to the masked out pitches get much less training than weights for other pitches and thus perform worse whenever they are actually used. For example, given a pitch that is masked out by subtracting 10^{38} (the value used in the MahlerNet implementation) from its corresponding logit only receives a very tiny, almost non-existing, gradient update whenever it is lower than the last pitch when a zero offset has been detected. Over time, this results in much less training for lower pitches and so they perform worse, lowering the overall performance. If this theory is true, then the correct way to handle a situation like this one is to mask out logits (or the ensuing softmax probabilities) only at inference time, and not at all during training, to avoid impeding the learning process. The same masking out is done for the zero duration which is forbidden whenever offset is zero but here, it only applies to a single duration class and it is thought to have relatively little impact on the training. Keeping three out of the four suggested improvements, one must keep in mind that these can only be used without scheduled sampling, since the input for these properties assume that the teacher-forced input will contain a certain sequence of events and with schedule sampling, this sequence is broken, rendering the inputs incorrect. This could be helped by turning the process of gathering these conditionings into a real-time process in the graph but this has not been done in the current version of MahlerNet. Thus, in the remaining experiments the scheduled sampling RNN decoder is not necessarily better than the plain RNN decoder which can make use of extra conditionings.

When running long runs modelling all properties, a slow learning curve was detected and so batch normalization was added. The addition of batch normalization changes the playing rules and, informally, a lot of setups were tested again with the verdict that LSTM units performed better than GRU units at all times. The new default setup was then to use LSTM units in all recurrent parts of MahlerNet and batch normalization in all feedforward layers (including the output layers) before the activation function. Furthermore, the activation function used was switched from tanh to leaky ReLU and, as indicated in the previous paragraph, extra conditionings with metric structure (beats), active pitches and instruments were used whenever possible. Even though training is facilitated by batch normalization, the results from the long runs still show that full reconstruction without teacher forcing is poor, with a large drop in accuracy for especially pitch. The suspicion here is that a so called "posterior collapse" takes place; a phenomenon discussed by the originators of the MusicVAE, among others ***(Bowman et al., 2015; Roberts et al., 2018; Roberts et al., 2017)***. The idea is that the decoder, being a powerful model in itself, learns to, in part, ignore the latent vector and base its decisions on the teacher-forced input instead. This results in good performance in training but poor reconstructions without teacher forcing. In this case, the use of scheduled sampling doesn't provide a solution either. In the first publication on MusicVAE, it is discussed how two bars of time slicing on sixteenth notes can be modelled in a flat decoder, but that for longer sequences, a hierarchical decoder must be used to improve the results ***(Roberts et al., 2017)***. Two bars of time slicing on sixteenth notes is equivalent to 32 events and given the distribution of sequence lengths in previous datasets compared to the sequence lengths in PIANOMIDI and MUSEDATA, it is somewhat expected that the threshold where a flat decoder, according to what is said by the originators of the MusicVAE, starts to degenerate lies somewhere in between. Thus, it seems that a full-scale solution to this problem must be alleviated by an architectural change not possible at this moment.

The long runs with VAE only resulted in models that could perform interpolations with varying results; endpoints with fewer events resulted in great interpolations but whenever the number of events in the endpoints was increased, the interpolation was less smooth and with only rudimentary characteristics of endpoints present throughout. The latter is an effect of the previously described posterior collapse and upon inspection, it is easy to see that the beginning of interpolated bars starts out well but then looses track. An implication of this is that interpolated bars often have reasonable bass notes, as these are modelled first in each moment of time, which creates a sense of a red thread and even though interpolations of endpoints with a large number of events are not smooth and ordered, the result is often musically interesting and brings about a groove-like feeling. Perhaps, the rudimentary repetition of the same interpolation sometimes gives the impression of repeated themes or motives, which is exactly the kind of structure that is hard to achieve. Thus, the interpolations can sometimes serve as compositions at the same time as it is a way to inspect the latent space. When random samples are drawn from the trained models, the results are catastrophic and lack any sense of musicality or musical idiomaticity; the notes are long, start on odd offsets and lie in extreme ranges. Since the VAE loss was made less significant in experiment 2, using the free bits and β-VAE techniques as was done in MusicVAE, the suspicion was that this had resulted in a latent space that was not smooth and that had a lot of unexplored holes which the decoders did not know what to do with. Running a test on the PIANOMIDI dataset with an RNN decoder with standard VAE loss confirmed this idea and the model could be seen generating musically plausible random samples but less impressive interpolations. Alas, in MusicVAE, a lot of focus is put on interpolations and so, even if the MusicVAE should have a latent space that is not smooth and does not allow for random samples, this is not a problem for that model. Nonetheless, the results shows clearly the tradeoff between a structured latent space and reconstruction accuracy that the originators of the MusicVAE write about. In MahlerNet, random samples drawn from the latent space is a large part of how the model is to be used and so it cannot be afforded to sacrifice the structure of the latent space on behalf of reconstruction accuracy; it is better with inexact but musically plausible reconstructions than gibberish random samples showing no musical structure whatsoever. Going the middle way and with one part left of experiment 2 and the whole experiment 3, it was decided that for models without context conditioning, it could be afforded to have a non-smooth and unpopulated latent space focused to give good interpolations and so the previously employed settings with β-VAE and free bits were used. On models conditioned on context however, the standard VAE loss would be employed instead.

Testing the BALSTM transpositional invariance property anew, it could again be seen that an RNN decoder performed better on a transposed dataset. However, an interesting thing here is that the BALSTM actually performs better than the RNN decoder when teacher forcing is employed. This is most likely due to the use of active pitches which creates a context for the BALSTM more similar to how it is used originally. Nevertheless, as could also be seen in other examples, the discrepancy between teacher-forced reconstruction and full reconstruction is now larger for the BALSTM decoder than for the RNN decoder and so full reconstruction is still better with a standard RNN decoder. At this point, it must be realized that the BALSTM brings nothing to the table and it was, with good reasons, discarded for further use as a decoder in the remaining parts of the experiments.

After training with the standard VAE loss and dropout, the resulting models with context conditioning did not reconstruct the training set with high accuracy, but instead short and long samples drawn from them often showed a high level of musical plausibility. It was apparent that they had learned basic musical properties from the training set, as opposed to details. In particular, for both the PIANOMIDI and MUSEDATA training sets, the models trained with scheduled sampling gave rise to slightly less convincing and more unstructured music. This indicates that the benefit of the added conditionings with beats and active pitches and instruments outshines the advantage of scheduled sampling, however, in a non-measurable way. Worth to mention here is also that there has been voices raised against scheduled sampling without questioning its empirical success ***(Huszár, 2015)*** and it is hard to know whether it is the scheduled sampling mechanism in itself that results in slightly poorer generated output or if it is the missed-out benefit from the additional conditionings. As with a lot of neural networks trained on music, long-term consistency is lacking but on a micro-level the music is full of examples related to the music the networks were trained on. When sampling from models trained on the MUSEDATA dataset, it is obvious that the set consists mostly of baroque and classic music with Bach playing a big role. In figure 5.8 for example, a typical baroque composition with continuo doubled in organ and cello with melody in flute and violin, sometimes playing together, at other times playing individually, is shown. The network manages to get a plausible setting of instruments and the level of counterpoint is high. Needless to say, models trained on the PIANOMIDI dataset almost exclusively chooses piano as instrument and nothing else. Also models trained on MUSEDATA show the tendency of wanting to stay with piano only once piano starts to show up in the score. Samples can be heard at the MahlerNet website at `https://www.mahlernet.se` but the list of musical qualities that can be found upon inspection is long:

- Music in the longer samples of 100 bars contains contrasting parts.
- Trumpet and timpani are often added in tutti sections, typical for use of brass and timpani in the classic era.
- Instruments most often play in their correct ranges.
- Beats 1 and 3 are often emphasized with longer notes and timpani beats, showing some understanding of meter. In particular, full bar notes are often placed at the first beat of the bar, no matter the meter which shows an understanding of meter.
- Instruments are doubled in realistic ways for the baroque and classic era.
- Typical instrument groups for baroque and classic music is prevalent with either the chamber setup or strings, woodwinds and timpani (less common with french horns however).
- Cadenzas and 5-to-1 motion is reoccurring even though to a lower extent than in reality for music from the baroque and classic era.
- Strings are the motor of the music whereas other instruments has the main focus only periodically and otherwise they contribute with shorter fills here and there.
- It is relatively common that whenever an instrument is activated, it plays a few phrases, not only contributing with a single sixteenth note and then going silent.

- Triplets are often accompanied by more triplets, thus the models do not change subdivision in an ad-hoc way. This is also a consequence of the context conditioning successively propagating the characteristics of previous music, bar by bar.

The context conditioning manages to create a red thread by linking pairs of bars together and the typical samples contain a continuously, but not rapidly, changing context in terms of register, active instruments, type of music (contrapuntal or homophonic as in baroque or classic music) and more. Samples are also diverse and often stay in style. These things clearly indicate that the context conditioning is used by the decoders and that it manages to create some long-term structure at certain levels, alas not in the level where clear musical themes are reoccurring, which is also somewhat expected given the very limited historical scope that it offers. Thus only tendencies and very high-level features are passed on in the generation of each new unit of music.

6.2.3 Experiment 3

Interpolations in models trained on the MAHLER dataset experience the same tendencies as those trained on PIANOMIDI and MUSEDATA, albeit with an even less smooth transition between end points in some cases since the MAHLER set has even longer sequences than the previous two datasets.

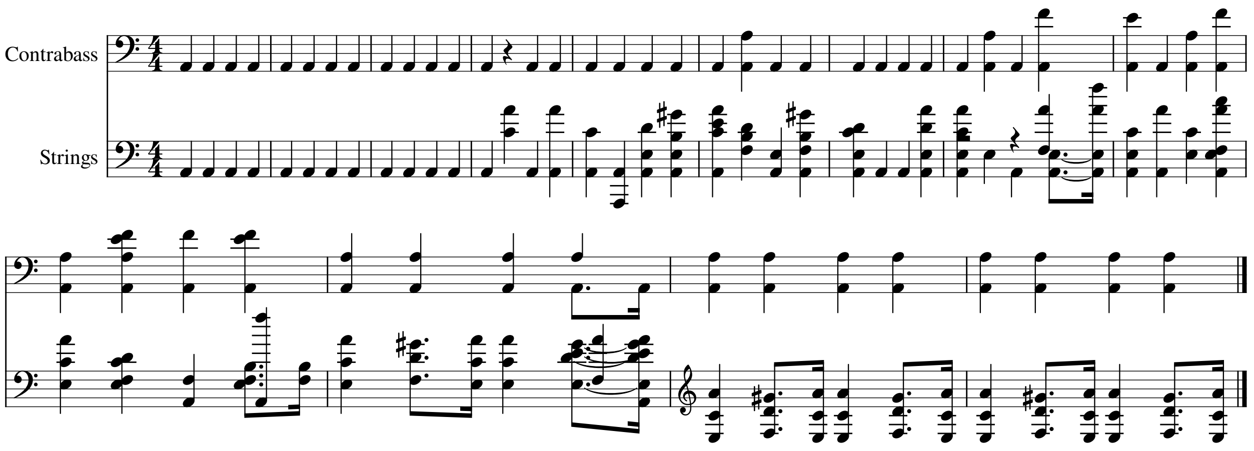
Seeded and unseeded random samples from models trained on the MAHLER dataset show widely different characteristics than the samples from experiment 2. Models trained on MUSEDATA had a tendency to go in to one of two modes: either a chamber setting with high polyphony or a homophonic setting with a typical classic sound. Noteworthy in experiment 3 is that no such chamber setting exists any longer and in general, instruments used are more romantic and the music is more dissonant. For example, tuba, trombone and trumpet rarely occurred in samples from experiment 2 but are frequently included in experiment 3. Tutti sections (where all or almost all instruments in the orchestra are playing) are commonly sampled which might be seen as a failure of the network. However, in reality, it is hard to know whether MahlerNet or Mahler himself is to blame; there is a reason to why some people think that the magnificent composer is too noisy as he often uses the full capacity of the orchestra. Other traits of romantic Mahler music, such as high strings, lines doubled in several woodwinds and mediant relationships between subsequent chords can also be spotted. Compared to experiment 2, music now proceeds in more unexpected directions harmonically (but typical for romantic music) and long sections dwelling on some (often dominant) chord with suspensions and temporary resolutions of dissonances are often heard. Sometimes, as stated by the originators of BachBot as well, pieces entirely generated by the model (no seed or in the case of BachBot, no initial voices to harmonize) sound better, perhaps because they lack the sometimes abrupt change of style when the seed transitions into MahlerNet's own composition. Because the harmonic turns are often unexpected, as is the case in a lot of romantic music, the samples may sound less pleasing than the samples from experiment 2. One view on this is that romantic music, especially that of Mahler, hinges more on themes and motives to tie together the music whereas both classic and baroque music are built on a somewhat sound harmonic structure. If this is the case, MahlerNet disfavours romantic music since it currently doesn't handle long-term structure in terms of motives and themes well enough. As in experiment 2, the model trained with scheduled sampling generates slightly less interesting and ordered music.

Revisiting the notion of posterior collapse and the MIDI format, it is common that tremolos or trills in strings create extremely long sequences of note repetitions in the MIDI files which of course disturbs the learning process and makes the sequences longer than they need to be, thus worsening the aspect of posterior collapse. In the written music however, a long tremolo or trill is usually written as a single note with an additional symbol.

6.3 On the Art of Training Neural Networks

It has already been mentioned previously that using one's intuition when constructing neural networks and the representations that they should work on might be a mistake, since it is hard to see the data from the perspective of the network and in some cases a representation that seems more complicated to a human, is actually easier to learn for a network. One such example was the question of using time slicing or not where time slicing seems to offer a simple scheme with a great built-in overview. Nonetheless, using less intuitive representations has often resulted in better results and this theme reappears in different areas with neural networks.

One such example is the latent space of the VAE. By measuring reconstruction capacity, we get an indication of what the model can accomplish, but it turns out that this is not always related to the landscape of the latent space, which becomes obvious when doing interpolations. First of all, evidently, even though batch normalization speeds up training it does not necessarily result in better full reconstruction capacity. In the diagram further down, all models perform similarly when reconstruction is done with teacher forcing, but without teacher forcing, there is quite a difference. More importantly, how does batch normalization affect the latent space? Even though there are models both with and without batch normalization among the best performers at full reconstruction in the diagram, the one thing that unites all models without batch normalization is that they, without exception, perform better interpolations in the latent space, posterior collapse aside.



Example 6.1: 10-step interpolation (sample 4-1) between similar bars (1 and 5) in Mahler61 from the MAHLER dataset. Model was trained with a VAE loss with 48 free bits and with single layers of 256 units in input encoder and decoder and a 128 unit large latent space.

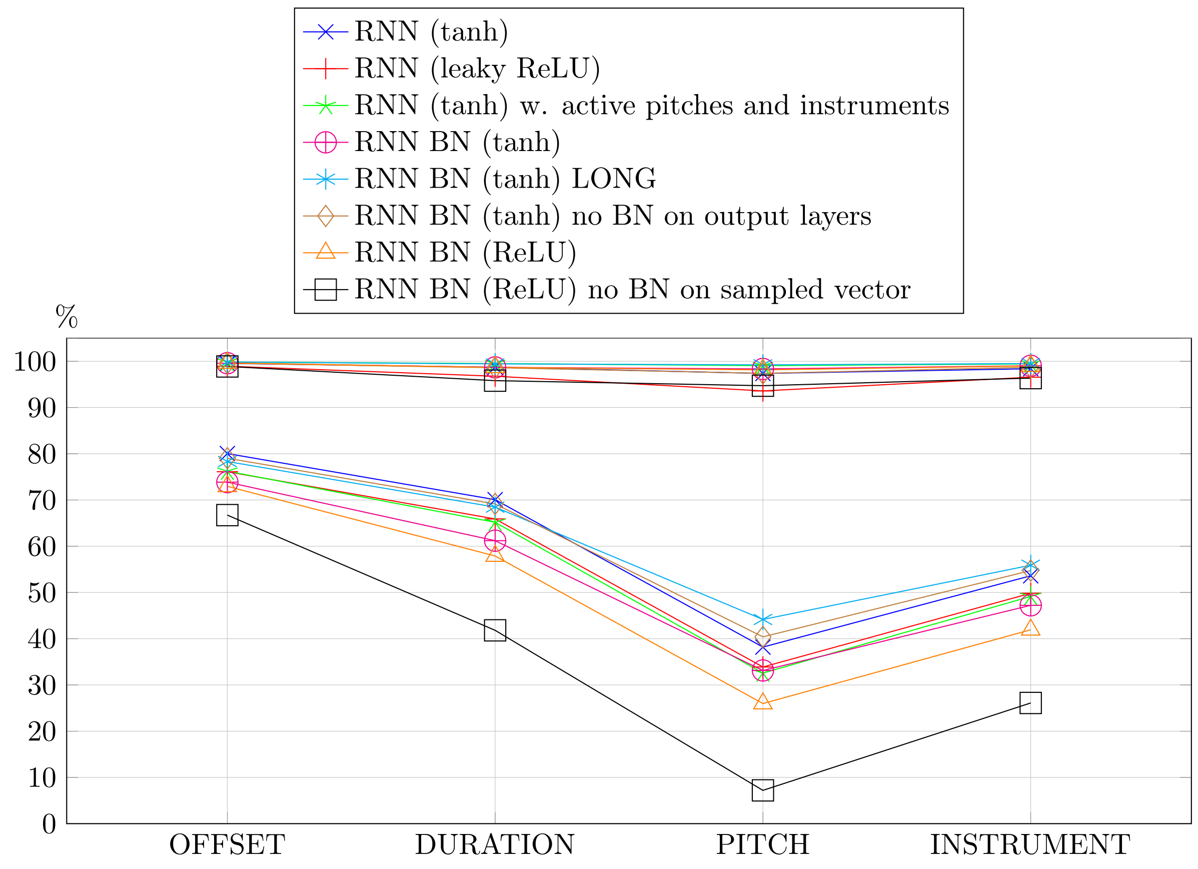
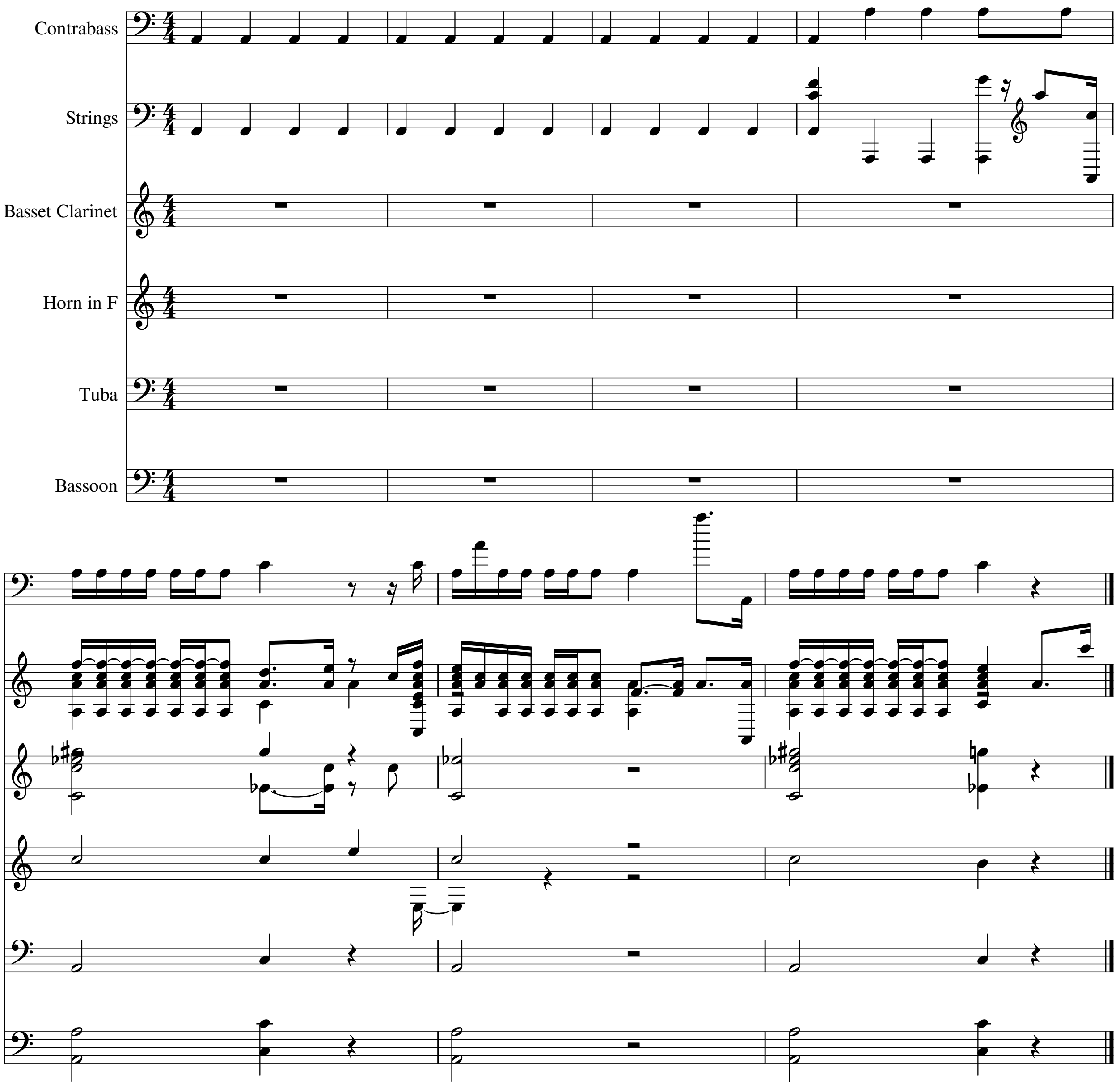


Figure 6.1: Comparisons between teacher-forced (upper) and full (lower) reconstruction of the dataset for models with only a VAE trained on a small subset of the MAHLER dataset with batch size 128, learning rate 0.001, RMSProp as optimizer and single layers with 256 units for both input encoder and decoder. The latent space had a size of 128 units and the beats conditioning was used for every model. Training was done over 440 epochs for all non-BN versions and 220 for models with BN except for the BN model marked with LONG which was also trained for 440 epochs. The free bits technique only was used in the VAE loss with 48 bits.

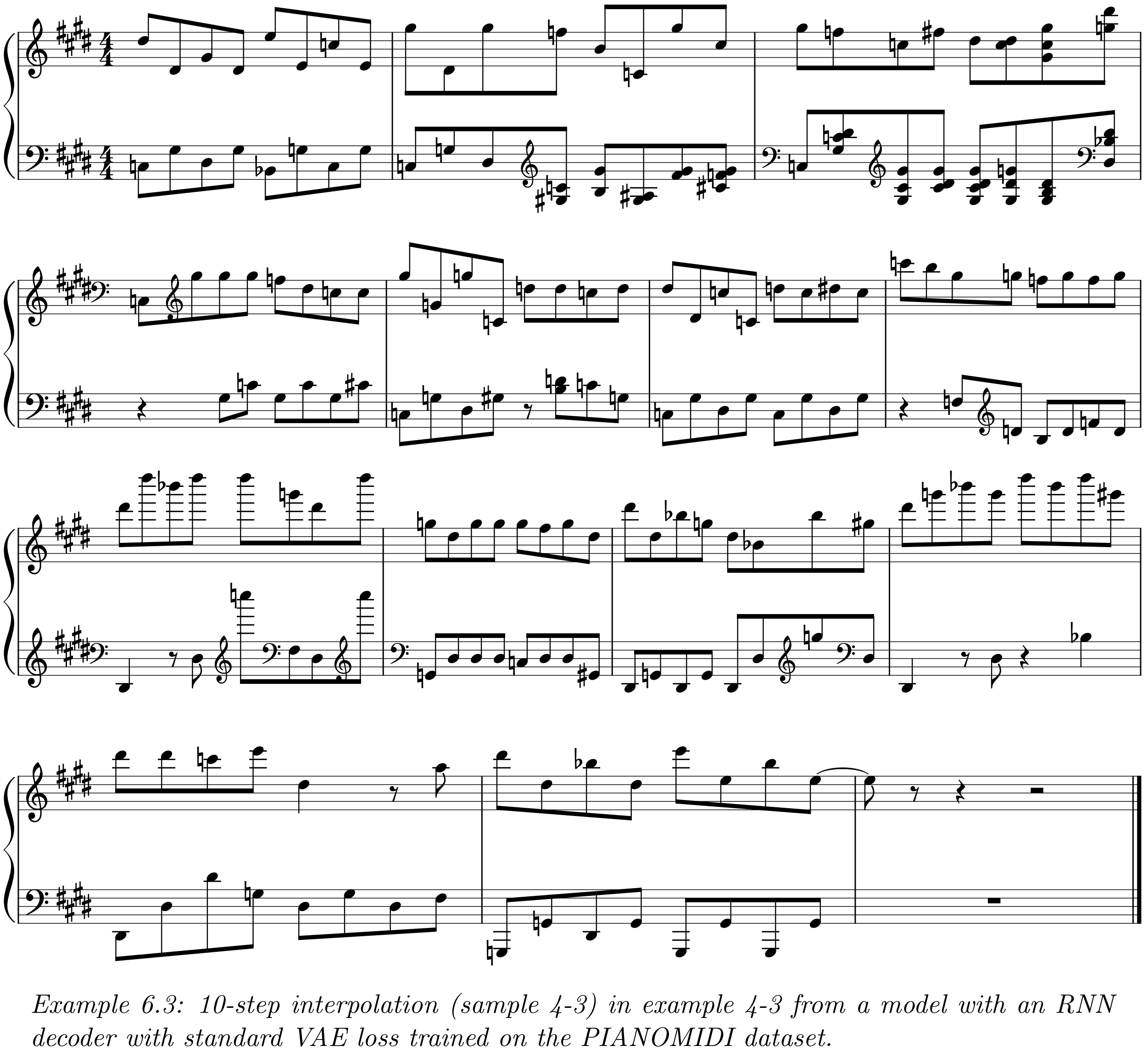
Example 6.1 shows an interpolation that is of high quality between two bars with moderately many events. The model used is simpler than many other models and not trained with batch normalization, but on the other hand with 48 free bits in the VAE loss, and still outperforms all the others. An even more impressive sample taken with the same model is presented below. Due to lack of space, only the first three and last two bars generated by the model are present (remember that the first and last bars are just copies off the original) and some instruments have been left out. The final bar contains a lot of events and the model produces very good results, alas the sign of posterior collapse shows towards the end of each bar where most mistakes are made.



Example 6.2: 10-step interpolation (sample 4-2) between similar bars (1 and 5) in Mahler61 from the MAHLER dataset. Model was trained with a VAE loss with 48 free bits and with single layers of 256 units in input encoder and decoder and a 128 unit large latent space.

There is thus a question aside from the posterior collapse that deals with how to measure the smoothness of the latent space and what means to use to get one. Batch normalization, in general, is useful without doubt but perhaps the rather particular scenario with a VAE causes it to have some negative side effects, perhaps dealing with the fact that training times are shortened which might result in less sampled vectors from the latent space which in turn might yield a latent space more prone to holes and irregularities.

As a final example in the discussion, a model that was not at all expected to perform well produced the below interpolation. It is not impressive in terms of an interpolation, because it is not very smooth nor exact. However, it has a certain musicality to it which further accentuates a lot of the questions of both philosophical and practical nature that are dealt with in algorithmic composition, manifested in this example with the following inquiry: is it an unsuccessful model because it cannot interpolate nicely between the endpoints, or is it a successful model because it can generate music that sounds nice?



Example 6.3: 10-step interpolation (sample 4-3) in example 4-3 from a model with an RNN decoder with standard VAE loss trained on the PIANOMIDI dataset.

7 FUTURE WORK

In this section future directions of work are discussed, initially with respect to MahlerNet. Zooming out, other strategies are then suggested, finally concluding with some remarks of the entire field of algorithmic composition with neural networks.

7.1 MahlerNet

MahlerNet has several bugs that should be fixed and improvements that can be made out of which a few stick out as more interesting to pursue:

- Support for the use of different size context and input should be implemented so that MahlerNet can be given a slightly larger contextual history to work with, making longer-term dependencies more likely.
- In orchestral music, trills and tremolos cause extra harm in MIDI since they are modelled as they sound, leading to an explosion of short notes which drastically increases the number of events in bars. Ideally, these should be modelled as they are notated and if this is not possible, they should with benefit be removed. An extra preprocessing step should therefore be constructed removing or simplifying these notes.
- The conversion to and from the low-level data representation should be handled by a plugin class to facilitate switching between data representations. This in order to, for example, attempt to use the same representation that is used in PerformanceRNN where only one event is modelled every time step (effectively collapsing the currently hierarchical output layers into a single output layer segmented into the different properties).
- More focus should be on voices in the input MIDI and less on instrument. For example, different string voices should be modelled as different instruments (nonetheless using the same actual output MIDI instrument) since merging all strings in the input training data into the same instrument creates exactly the kind of unidiomatic music that MahlerNet should not generate. As a consequence, such patterns are often spotted in the output strings.
- To further facilitate long-term structure, the context can be made more dependent on the previous context by storing the RNN states in between contexts. This makes it necessary to process data in a certain order however and also requires some extra preprocessing steps.

- Given that the standard RNN performed better than the BALSTM, a new approach should be attempted where each training sample is randomly transposed to a nearby key, as is done in MusicVAE.
- Focusing more on the structure of the latent space, t-sne visualizations should be implemented to further investigate how and when for example batch normalization has a negative impact.
- Weighed losses should be implemented to place a higher priority on pitch which is harder to model than the remaining properties.
- Investigate a loss component that is proportional to the distance between target class and predicted class; both offset, duration and pitch classes are modelled in an ordered way so that the closer to the correct class the prediction is, the better a model you have.
- Additional conditioning regarding for example composer, music genre and time signature should be introduced and evaluated.

On a side note, exchanging the Mido library for another one, might also prove beneficial since Mido turned out to be unable to parse quite a lot of MIDI files, even from well-known datasets.

7.2 Other Strategies

During and after the work of MahlerNet, a number of other similar strategies have been thought of and should ideally be evaluated as well:

- Implementing the concepts of MahlerNet as part of a recurrent variational network, similar to how the VRNN works ***(Chung et al., 2015)***, so that only one time step at a time is modelled, allowing for long-term dependencies of, in theory, arbitrary length. There have been a few attempts similar to this but none of them uses the same layout as the VRNN which is the variational recurrent model that have departed the least from a standard RNN.
- Model each output track, voice or instrument independently with a feedforward network modelling what tracks (or voices or instruments) to include. This would make some assumptions about the modelled properties and how they correlate between time steps more correct. Tracks or instrument information from MIDI files could be used for this purpose during training. If the needed information is not present in the input files, extra annotations would be required.
- Decode each property in an independent RNN. This does not make sense intuitively but would still be interesting to investigate.
- Implement a hierarchical model where the high hierarchy submodel outputs a representation for every non-zero offset that is moved ahead and a lower hierarchy submodel is responsible for modelling all the simultaneous output at that time (allowing for sequential polyphony in

the lower hierarchy submodel). This could potentially solve the problem with mode collapse as well and resembles the MusicVAE, albeit with a different hierarchical modelling strategy.

7.3 Future Directions and Conceptual Ideas

Some developments and trends in machine learning seem more appropriate for music and are likely legitimate indicators of which directions to head down. Given that the output music should be in the same style as some music that exists today, it should contain repetitions, e.g. express self-similarity, and at the same time give room for contrasts. These properties, not necessarily similar to any other sequential modelling field, lead to the notions of hierarchies and attention becoming very interesting and appropriate. Hierarchies because music is hierarchical and is built up from sections, passages, phrases, themes and motives, and attention because music is self-similar and refers to previous parts of itself. A common musico-philosophical thought, formalized by Lerdahl and Jackendoff in a Generative Theory of Tonal Music ***(Lerdahl and Jackendoff, 1982)*** is the concept of musical patterns and structures being perceived and constructed by the human listening to music, as opposed to necessarily being present in the material. An ensuing thought in this direction would then be to not bother with any guidance or particular measures at all since we will find some patterns to hang on to no matter. Even though the observation by Lerdahl and Jackendoff is likely, our brains have limits and there must be some sort of patterns for our brains to be able to detect it, and so the notion of self-similarity is very central to the concept of music, no matter if it is a property welded into the music on purpose or not. An informal proof of that not all sounds contain patterns to be found is that there exists something that we refer to as noise which, essentially, is unstructured sounds without patterns.

In some areas, it is harder to determine the future developments. For example, how the trend to use networks with several more or less disjoint components doing different things will stand against the monolithic approach of one end-to-end network doing all the work is hard to say. The same goes for the approach that networks are merely a tool for the composer who guides the network, or uses its output as ideas or partial compositions to be finalized by herself, that stand against the dream of models that output fully-fledged pieces of music exhibiting all the properties of qualitative music that we might have. In these areas, it is thus a good idea to stick with a variety of approaches and see what direction future research takes.

As far as training data is concerned, the field of algorithmic composition of symbolic music with neural networks suffers from a barrier holding it back. There are tons of sheet music available online but as of today, there is no simple and efficient way to convert it to a data representation. Conversions to MIDI have been done to a large extent but after working with MahlerNet, it is obvious that MIDI does not map onto written music and introduces a number of problems that moves the data representation away from the symbolic musical domain in the direction towards the sounding musical domain. Therefore the MIDI format does not fill the intended purpose to bridge the aforementioned gap. Formats like ABC and MusicXML are suitable since they have a one-to-one relationship to symbolic music but these formats are not as widely spread and used as is MIDI. An important contribution that needs to be investigated would therefore be computer vision algorithms that can take a PDF file of written music and turn it into an exact data representation, effectively making preprocessing and therefore learning much more efficient and with more data available. Such software does exist today, but to the knowledge of the author, it seems as if it only works under very special circumstances and not at all for most of the scanned sheet music available on the web today. A negative side effect of the excessive amount of preprocessing of MIDIs is that it is harder to compare publications, because there might be important differences on a micro level in the preprocessing not accounted for in writing. With a more uniform way to represent symbolic music as data, this problem would, at least in part, go away. Alas, the data representation and training data is of utter importance as previously remarked. More unified standards as far as evaluation is concerned is also desirable for the future as is also a closer connection to the real-world field where algorithms are intended to play a role, as discussed in the summary in chapter 3. Hence samples should always be available, without exception, for contributions to matter.

On a more artistic notion, going in an entirely other direction than earlier, algorithmic composition is also used to create new music that does not mimic human compositions, and for the future, it is desirable that this direction is explored to a higher degree. After all, one might with reason object against having access to the unlimited boundaries of a computer and still attempting to do what has been done manually for about a thousand years. This path of exploration could involve microtones, quarter tones, new scales or entirely new sounds and would really amount to a new type of composition with new limitations and horizons. One such example is when Barlow uses the computer to create music that has fourteen different tempi in parallel ***(Supper, 2001)***.

At the time of the finalization of this work, a new model from OpenAI named MuseNet has just been disclosed and the preliminary glimpses reveal that it is an absolutely extraordinary model, advancing the field of algorithmic composition with neural networks a great deal. The remarkable results are among the first surfacing from using the relatively new Transformer architecture, in which sequence to sequence models are built using feedforward networks with attention only, rendering the recurrent parts of the the traditional sequence to sequence network obsolete. Among other things, such a model has the advantages over RNNs that it lends itself to parallelization and also maintains a shorter path in the computational graph between input and target. MuseNet has more than seventy layers and uses a context of over 4000 tokens to predict the next; a task that it manages exemplary, effectively establishing the Transformer as the new state of the art, and subsequently the future, in the field.

8 CONCLUSION

Music is an art, which makes it harder to determine which way to go and how to weigh results, when compared to the development in for example image classification, where right and wrong are clear and distinct. This has caused the field of algorithmic composition with neural networks to be diverse, and it is of great joy to see the creativity of composing music get shifted to the creativity of modelling a network that composes music.

In this work, an elaborate contemporary literature survey over algorithmic composition in general and algorithmic composition with neural networks in particular has been presented. Inspired by previous work and some of the more successful achievements in the field, MahlerNet, a new neural network architecture adhering to a predefined set of conditions out of which all were fulfilled, has been suggested. The novelty of MahlerNet lies in how existing components of neural networks used in algorithmic composition have been put together as well as the preprocessing steps used to transform MIDI to a data representation with symbolic music in mind. The possibility to model an arbitrary number of instruments, in this work restricted to 23, is also, to the author's best knowledge, a new contribution to the community as is therefore also the data representation used even though some parts of it have been used before. MahlerNet was evaluated with a baseline RNN decoder and a BALSTM decoder out of which the former turned out to be superior. The BALSTM decoder on the other hand is slower to train, requires more memory and performs worse than the baseline RNN decoder, even when evaluated for pitch invariance; the very property it was picked for. Reasons for the BALSTM performing bad might be that the context in which it is deployed differs significantly from how it was originally used, but additional investigation is needed to be sure. It was furthermore concluded that even though seemingly contributing positively, both batch normalization and scheduled sampling gave rise to results that were worse than with the vanilla setup. Batch normalization especially seems to affect the latent space negatively even though more research is needed to support this claim. Scheduled sampling improved, if anything, training but subjectively resulted in slightly less interesting samples and could only be used interchangeably with some extra conditionings, that was not possible to use along with scheduled sampling. Even though there has been some criticism against scheduled sampling, it is hard to know whether the subtle advantage of not using it here comes from the extra conditionings or is connected to the criticism in question. Finally, the VAE free bits and β-VAE extensions had a bad impact on the latent space which could most obviously be determined by very poor random sample quality; a feature not obvious from the MusicVAE, one of the MahlerNet predecessors. MahlerNet honors all the conditions that was placed on it and lends itself to arbitrary further conditionings, for example with information about genre, composer or time signature.

MahlerNet manages to generate musically idiomatic output which bears significant traits from the music it has been trained on. The historically limited scope of one bar of context conditions MahlerNet and creates long-term structure between bars of music in terms of instrumentation, dynamics, basic patterns and key. There is however little long-term structure as far as more concrete musical building blocks, such as motives and themes, are concerned. When applied to bars with longer sequences of music, MahlerNet suffers from posterior collapse and has trouble to reconstruct the input without teacher forcing. The former lack of long-term structure is understandable whereas the latter indicates a somewhat flawed architecture which calls for a change in design. MahlerNet generates interesting music, perhaps in different ways, both from predicting the next bar based on the previous and by means of interpolation between two bars. The latter does not always result in a smooth interpolation, but instead often the music is interesting and with a thread, in terms of repeated bass notes or variations on a theme, that might be easier to follow than the free compositions. Interpolations and samples from the latent space of the VAE show that structure, density and smoothness of this space depend on more than just full reconstruction accuracy, alas more research has to determine exactly what constitutes favorable training for the latent space to possess these advantageous characteristics. Often, music generated from scratch, as opposed to seeded, is more coherent even though MahlerNet often manages to continue in somewhat the same style when generation is started with a seed. Even though all generated music has qualitative aspects and resemble the music used during training, models trained on baroque and classic music tend to produce music that sounds better than the models trained on romantic Mahler music. Reasons for this may be that romantic music in general is more dissonant and messy, especially Mahler, than baroque and classic music but in particular, romantic music is often held together by the use of motives and themes which is exactly the type of long-term structure that MahlerNet cannot handle.

MahlerNet can be improved and further explored in several ways and future models, preferably based on hierarchies and attention, will continue to improve results but the most important future development for modelling music with neural networks is to bridge the gap between the enormous amounts of symbolic music that exist in sheet music form and some representation that can work as input to a neural network pipeline. This could be solved by an increased interest in, and subsequent standardization of, one of the existing formats that maps more directly to written symbolic music than does MIDI. These formats have existed for a long time and since no informal consensus, as to what format to widely use, exists, a sudden change in this aspect is highly unlikely. The most important way forward is therefore to develop computer vision techniques for reading arbitrary scores and sheet music into a data representation that adheres more to the written representation than does MIDI.

ACKNOWLEDGEMENTS

The author of this thesis would like to thank Bob Sturm for having been an available and insightful supervisor.

The following software was used during the work:

- NotationPlayer3 - software to render MIDI files into written music
- Musescore - software to render MIDI files into written music
- Audacity - audio editing software
- PDFResizer.com - online PDF tool where you can crop your PDFs
- Overleaf.com - online LaTeX tool for writing and printing to PDF

BIBLIOGRAPHY

Adiloglu, K. and Alpaslan, F. N. (2007). A machine learning approach to two-voice counterpoint composition. *Knowledge-Based Systems*, 20:300–309.

Agarwala, N., Inoue, Y., and Sly, A. (2017). Music Composition using Recurrent Neural Networks. Technical report, Stanford University.

Baggi, D. L. (1998). Università Degli Studi di Milano "The Role of Computer Technology in Music and Musicology". `http://www.lim.di.unimi.it/events/ctama/baggi.htm`. Accessed: 2018-03-03.

Bayer, J. and Osendorfer, C. (2015). Learning Stochastic Recurrent Networks. In *International Conference on Learning Representations 2015*.

Bayer, J., Osendorfer, C., Urban, S., Chen, N., Korhammer, D., and van der Smagt, P. (2013). On Fast Dropout and its Applicability to Recurrent Networks. In *ICLR 2014*, Calgary, Canada.

Bellgard, M. I. and Tsang, C. P. (1994). Harmonizing Music the Boltzmann Way. *Connection Science*, pages 281–297.

Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. (2015). Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks. *CoRR*, abs/1506.03099.

Bengio, Y., Boulanger-Lewandowski, N., and Pascanu, R. (2012). Advances in Optimizing Recurrent Networks. *CoRR*, abs/1212.0901.

Berger, J. and Gang, D. (1997). A Neural Network Model of Metric Perception and Cognition in the Audition of Functional Tonal Music. *1998 International Computer Music Conference*.

Berger, J. and Gang, D. (1999). A Real Time Model of the Formulation and Realization of Musical Expectations. Submitted to Musical Perception.

Bickerman, G., Bosley, S., Swire, P., and Keller, R. (2010). Learning to Create Jazz Melodies Using Deep Belief Nets. In *First International Conference on Computational Creativity*.

Boulanger-Lewandowski, N., Bengio, Y., and Vincent, P. (2012). Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription. *Proceedings of the 29th International Conference on Machine Learning (ICML 2012)*.

Bowman, S. R., Vilnis, L., Vinyals, O., Dai, A. M., Józefowicz, R., and Bengio, S. (2015). Generating Sentences from a Continuous Space. *CoRR*, abs/1511.06349.

Bretan, M., Oore, S., Eck, D., and Heck, L. P. (2017). Learning and Evaluating Musical Features with Deep Autoencoders. *CoRR*, abs/1706.04486.

Bretan, M., Weinberg, G., and Heck, L. (2016). A Unit Selection Methodology for Music Generation Using Deep Neural Networks. *CoRR*.

Briot, J., Hadjeres, G., and Pachet, F. (2017). Deep Learning Techniques for Music Generation - A Survey. *CoRR*, abs/1709.01620.

Brunner, G., Wang, Y., Wattenhofer, R., and Wiesendanger, J. (2017). JamBot: Music Theory Aware Chord Based Generation of Polyphonic Music with LSTMs. *CoRR*, abs/1711.07682.

Cage, J. (1961). *Silence: Lectures and Writings by John Cage*. University Press of New England, Hannover, New Hampshire, USA, 2nd edition.

Chen, C.-C. J. and Miikkulainen, R. (2001). Creating Melodies with Evolving Recurrent Neural Networks. *Proceedings of the 2001 International Joint Conference on Neural Networks*, pages 2241–2246.

Chen, Z., Wuy, C.-W., Yen-Cheng, L., Lerchy, A., and Chang-Tien, L. (2017). Learning to Fuse Music Genres with Generative Adversarial Dual Learning. *2017 IEEE International Conference on Data Mining*, pages 817–822.

Choi, K., Fazekas, G., and Sandler, M. B. (2016). Text-based LSTM networks for Automatic Music Composition. *CoRR*, abs/1604.05358.

Chu, H. C., Urtasun, R., and Fidler, S. (2016). Song From PI: A Musically Plausible Network for Pop Music Generation. *CoRR*, abs/1611.03477.

Chung, J., Gülçehre, Ç., Cho, K., and Bengio, Y. (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *CoRR*, abs/1412.3555.

Chung, J., Kastner, K., Dinh, L., Goel, K., Courville, A. C., and Bengio, Y. (2015). A Recurrent Latent Variable Model for Sequential Data. *CoRR*, abs/1506.02216.

Coca, A. E., Romero, R. A. F., and Zhao, L. (2011). Generation of composed musical structures through recurrent neural networks based on chaotic inspiration. *Proceedings of International Joint Conference on Neural Networks*, pages 3220–3226.

Colombo, F. and Gerstner, W. (2018). BachProp: Learning to Compose Music in Multiple Styles. *CoRR*, abs/1802.05162.

Colombo, F., Muscinelli, S., Seeholzer, A., Brea, J., and Gerstner, W. (2016). Algorithmic Composition of Melodies with Deep Recurrent Neural Networks. In *1st Conference on Computer Simulation of Musical Creativity*.

Colombo, F., Seeholzer, A., and Gerstner, W. (2017). Deep Artificial Composer: A Creative Neural Network Model for Automated Melody Generation. In *Computational Intelligence in Music, Sound, Art and Design*, pages 81–96, Cham. Springer International Publishing.

Corrêa, D. C., Levada, A. L. M., Saito, J. H., and Mari, J. F. (2008). Neural network based systems for computer-aided musical composition: supervised x unsupervised learning. *Proceedings of the 2008 ACM symposium on Applied computing*, pages 1738–1742.

De Felice, C., De Prisco, R., Malandrino, D., Zaccagnino, G., Zaccagnino, R., and Zizza, R. (2015). Chorale music splicing system: An algorithmic music composer inspired by molecular splicing. In *EvoMUSART2015, Evolutionary and Biologically Inspired Music, Sound, Art and Design*, pages 50–61, Cham. Springer.

De Felice, C., De Prisco, R., Malandrino, D., Zaccagnino, G., Zaccagnino, R., and Zizza, R. (2017). Splicing Music Composition. *Inf. Sci.*, 385:196–212.

De Prisco, R., Malandrino, D., Zaccagnino, G., Zaccagnino, R., and Zizza, R. (2017). Chorale music splicing system: An algorithmic music composer inspired by molecular splicing. In *Evo-MUSART 2017, Computational Intelligence in Music, Sound, Art and Design*, pages 97–113, Cham. Springer.

Diaz-Jerez, G. (2011). Composing with Melomics: Delving into the Computational World for Musical Inspiration. *Leonardo Music Journal*, 21:13–14.

Dong, H.-W., Hsiao, W.-Y., Yang, L.-C., and Yang, Y.-H. (2017). MuseGAN: Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment. *AAAI 2018*.

Duff, M. O. (1989). Backpropagation and Bach's 5th cello suite (Sarabande). *Proceedings of the International Joint Conference on Neural Networks*, page 575.

Eck, D. and LaPalme, J. (2006). Learning Musical Structure Directly from Sequences of Music. Technical report, Université de Montréal.

Eck, D. and Schmidhuber, J. (2002). A First Look at Music Composition using LSTM Recurrent Neural Networks. *IDSIA Technical Report*.

Edwards, M. (2011). Algorithmic Composition: Computational Thinking in Music. *Communications of the ACM*, 54(7):58–67.

Engel, J., Resnick, C., Roberts, A., Dieleman, S., Eck, D., Simonyan, K., and Norouzi, M. (2017). Neural Audio Synthesis of Musical Notes with WaveNet Autoencoders. *CoRR*, abs/1704.01279.

Eppe, M., Alpay, T., and Wermter, S. (2018). Towards End-to-End Raw Audio Music Synthesis. In *Proceedings of the 27th Conference on Artificial Neural Networks (ICANN)*.

Fabius, O. and van Amersfoort, J. R. (2015). Variational Recurrent Auto-Encoders. *ICLR 2015*.

Fernàndez, J. D. and Vico, F. (2013). AI Methods in Algorithmic Composition: A Comprehensive Survey. *Journal of Artificial Intelligence*, 48:513–582.

Franklin, J. A. (2001). Multi-Phase Learning for Jazz Improvisation and Interaction. *In Proc. Eighth Biennial Symposium on Arts and Technology.*

Franklin, J. A. (2004). Recurrent Neural Networks and Pitch Representations for Music Tasks. *Proc. 2004 Florida AI Research Sympos. (FLAIRS04) Special Track on AI and Music.*, pages 33–37.

Franklin, J. A. (2005). Jazz Melody Generation from Recurrent Network Learning of Several Human Melodies. *Proceedings of the Eighteenth International Florida Artificial Intelligence Research Society Conference.*

Franklin, J. A. (2005 / 2006). Recurrent Neural Networks for Music Computation. *Informs Journal on Computing*, pages 321–338.

Franklin, J. A. and Locke, K. K. (2004 / 2005). Recurrent Neural Networks for Musical Pitch Memory and Classification. *International Journal on Artificial Intelligence Tools*, pages 329–342.

Freisleben, B. (1992). The Neural Composer: A Network for Musical Applications. *Proceedings of the 1992 International Conference on Artificial Neural Networks*, pages 1663–1666.

Gan, Z., Li, C., Henao, R., Carlson, D., and Carin, L. (2015). Deep Temporal Sigmoid Belief Networks for Sequence Modeling. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pages 2467–2475, Cambridge, MA, USA. MIT Press.

Gang, D. and Berger, J. (1996). Modeling the Degree of Realized Expectation in Functional Tonal Music: A Study of Perceptual and Cognitive Modeling Using Neural Networks. In *Proceedings of the International Computer Music Conference.*

Gang, D., Lehmann, D., and Wagner, N. (1999). Tuning Neural Network for Harmonizing Melodies in Real-Time. In *International Computer Music Conference (ICMC98).*

Gang, D. and Lehmann, D. J. (1995). An Artificial Neural Net for Harmonizing Melodies. In *ICMC.*

Gang, D., Lehmann, D. J., and Wagner, N. (1997). Harmonizing Melodies in Real-Time: the Connectionist Approach. In *Proceedings of the International Computer Music Conference.*

Github.com (2019). MIDI Objects for Python. `https://github.com/mido/mido`. Accessed: 2019-05-15.

Goel, K., Vohra, R., and K., S. J. (2014). Polyphonic Music Generation by Modeling Temporal Dependencies Using a RNN-DBN. *Lecture Notes in Computer Science*, 8681:217–224.

Goldman, C. V., Gang, D., and S., R. J. (1996 / 1999). NetNeg: A connectionist-agent integrated system for representing musical knowledge. *Annals of Mathematics and Artificial Intelligence*, pages 69–90.

Goodfellow, I. J. (2017). NIPS 2016 Tutorial: Generative Adversarial Networks. *CoRR*, abs/1701.00160.

Goodfellow, I. J., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. (2013). Maxout Networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13.

Gregor, K., Danihelka, I., Graves, A., and Wierstra, D. (2015). DRAW: A Recurrent Neural Network For Image Generation. *CoRR*, abs/1502.04623.

Guimaraes, G., Sanchez-Lengeling, B., Outeiral, C., Cunha Farias, P. L., and Aspuru-Guzik, A. (2017). Objective-Reinforced Generative Adversarial Networks (ORGAN) for Sequence Generation Models. *CoRR*.

Gülçehre, Ç., Cho, K., Pascanu, R., and Bengio, Y. (2013). Learned-norm pooling for deep neural networks. *CoRR*, abs/1311.1780.

Hadjeres, G. and Nielsen, F. (2017). Interactive Music Generation with Positional Constraints using Anticipation-RNNs. *CoRR*, abs/1709.06404.

Hadjeres, G., Nielsen, F., and Pachet, F. (2017). GLSR-VAE: Geodesic Latent Space Regularization for Variational AutoEncoder Architectures. *CoRR*, abs/1707.04588.

Hadjeres, G. and Pachet, F. (2017). DeepBach: a Steerable Model for Bach chorales generation. *CoRR*, abs/1612.01010.

Hennig, J. A., Umakantha, A., and Williamson, R. C. (2017). A Classifying Variational Autoencoder with Application to Polyphonic Music Generation. *CoRR*, abs/1711.07050.

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A. (2016). β-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. In *Proceedings of the Fifth International Conference on Learning Representations*, ICLR 2017.

Hild, H., Feulner, J., and Menzel, W. (1991). HARMONET: a neural net for harmonising chorales in the style of J.S. Bach. *Advances in Neural Information Processing 4*, pages 267–274.

Hiller, L. A. and Isaacson, L. M. (1959). *Experimental Music: Composition with an Electronic Computer*. McGraw-Hill Book Company, 1st edition.

Huang, A. and Wu, R. (2016). Deep Learning for Music. *CoRR*, abs/1606.04930.

Huang, C.-Z., Cooijmans, T., Roberts, A., Courville, A., and Eck, D. (2017). Counterpoint by Convolution. In *The 18th International Society for Music Information Retrieval Conference*.

Huang, C.-Z. A., Duvenaud, D., and Gajos, K. Z. (2016). ChordRipple: Recommending Chords to Help Novice Composers Go Beyond the Ordinary. In *Proceedings of the 21st International Conference on Intelligent User Interfaces*, IUI '16, pages 241–250, New York, NY, USA. ACM.

Huszár, F. (2015). A Word of Caution on Scheduled Sampling for Training RNNs. `https://www.inference.vc/scheduled-sampling-for-rnns-scoring-rule-interpretation/`. Accessed: 2019-06-11.

Hörnel, D. and Menzel, W. (1998). Learning Musical Structure and Style with Neural Networks. *Computer Music Journal*, 22(4):44–62.

Jacob, B. L. (1996). Algorithmic Composition As a Model of Creativity. *Org. Sound*, 1(3):157–165.

Jaques, N., Gu, S., Bahdanau, D., Hernández-Lobato, J. M., Turner, R. E., and Eck, D. (2017). Sequence Tutor: Conservative Fine-Tuning of Sequence Generation Models with KL-control. *CoRR*.

Jaques, N., Gu, S., Bahdanau, D., Lobato, J. M. H., Turner, R. E., and Eck, D. (2016). Tuning Recurrent Neural Networks With Reinforcement Learning. In *NIPS 2016 Deep Reinforcement Learning Workshop*.

Johnson, D. D. (2017). Generating Polyphonic Music Using Tied Parallel Networks. In *EvoMusArt2017*.

Kalingeri, V. and Grandhe, S. (2016). Music Generation with Deep Learning. *CoRR*, abs/1612.04928.

Kingma, D. P., Salimans, T., and Welling, M. (2016). Improving Variational Inference with Inverse Autoregressive Flow. *CoRR*, abs/1606.04934.

Koh, E. S., Wright, D., and Dubnov, S. (2018). Capturing Musical Structure Using Convolutional Recurrent Latent Variable Model. Submitted to ICLR2018.

Koutník, J., Greff, K., Gomez, F. J., and Schmidhuber, J. (2014). A Clockwork RNN. *CoRR*, abs/1402.3511.

Laden, B. and Keefe, D. H. (1989). The Representation of Pitch in a Neural Net Model of Chord Classification. *Computer Music Journal*, 13(4):12–26.

Large, E. W., Palmer, C., and Pollack, J. B. (1995). Reduced Memory Representations for Music. *Cognitive Science*, 19:53–96.

Lattner, S., Grachten, M., and Widmer, G. (2016). Imposing higher-level Structure in Polyphonic Music Generation using Convolutional Restricted Boltzmann Machines and Constraints. *CoRR*, abs/1612.04742.

Lee, S.-g., Hwang, U., Min, S., and Yoon, S. (2017). A SeqGAN for Polyphonic Music Generation. *CoRR*.

Lerdahl, F. and Jackendoff, R. S. (1982). *A Generative Theory of Tonal Music*. The MIT Press, Cambridge, MA, USA, 1st edition.

Lewis, J. (1991). *Music and Connectionism*, chapter Creation by Refinement and the Problem of Algorithmic Music Composition, pages 212–228. The MIT Press.

Liang, F., Gotham, M., Johnson, M., and Shotton, J. (2017). Automatic Stylistic Composition of Bach Chorales with Deep LSTM. In *18th International Society for Music Information Retrieval Conference*.

Lindenmayer, A. (1968a). Mathematical Models for Cellular Interactions in Development: I. Filaments with One-sided Inputs. *Journal of Theoretical Biology*, pages 280–299.

Lindenmayer, A. (1968b). Mathematical Models for Cellular Interactions in Development: II. Simple and Branching Filaments with Two-sided Inputs. *Journal of Theoretical Biology*, pages 300–315.

Lischka, C. (1987). Connectionist Models of Musical Thinking. *Proceedings of the International Computer Music Conference 1987*, pages 190–196.

Lischka, C. (1989 / 1991). Understanding Music Cognition: A Connectionist View. *Representations of Musical Signals*, pages 417–445.

Liu, I.-T. and Ramakrishnan, B. (2014). Bach in 2014: Music Composition with Recurrent Neural Network. *Under review as a workshop contribution at ICLR 2015.*

Lousseief, E. (2015). *Med motiv som tema*. Globe Edit, Saarbrücken, Germany, 1st edition.

Lyu, Q., Wu, Z., Zhu, J., and Meng, H. (2015). Modelling High-dimensional Sequences with LSTM-RTRBM: Application to Polyphonic Music Generation. In *Proceedings of the 24th International Conference on Artificial Intelligence*.

Madjiheurem, S., Qu, L., and Walder, C. (2016). Chord2Vec: Learning Musical Chord Embeddings. In *Constructive Machine Learning 2016*.

Malik, I. and Ek, C. H. (2017). Neural Translation of Musical Style. *CoRR*, abs/1708.03535.

Mao, H. H., Shin, T., and Cottrell, G. W. (2018). DeepJ: Style-Specific Music Generation. *CoRR*, abs/1801.00887.

Mehri, S., Kumar, K., Gulrajani, I., Kumar, R., Jain, S., Sotelo, J., Courville, A. C., and Bengio, Y. (2017). SampleRNN: An Unconditional End-to-End Neural Audio Generation Model. *CoRR*, abs/1612.07837.

Melo, A. F. and Wiggins, G. A. (2003). A Connectionist Approach to Driving Chord Progressions Using Tension. *Proceedings of the AISB '03 Symposium on Creativity in Arts and Science.*

Meyer, L. B. (1967). *Music, the Arts, and Ideas: Patterns and Predictions in Twentieth-century culture.* The University of Chicago Press, Chicago, AZ, USA, 2nd edition. Accessed: 2018-03-02.

Mogren, O. (2016). C-RNN-GAN: Continuous Recurrent Neural Networks with Adversarial Training. *CoRR.*

Mordvintsev, A., Olah, C., and Tyka, M. (2015). Inceptionism: Going Deeper into Neural Networks. `https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html`.

Mozer, M. C. (1990). Connectionist Music Composition Based on Melodic, Stylistic, and Psychophysical Constraints. *Computer Science Technical Reports.*

Mozer, M. C. (1994). Neural Network Music Composition by Prediction: Exploring the Benefits of Psychoacoustic Constraints and Multi-Scale Processing. *Connection Science*, pages 247–280.

Mozer, M. C. and Soukup, T. (1991). Connectionist Music Composition Based on Melodic and Stylistic Constraints. *Advances in Neural Information Processing 3*, pages 789–796.

Nayebi, A. and Vitelli, M. (2015). GRUV : Algorithmic Music Generation using Recurrent Neural Networks.

Nierhaus, G. (2009). *Algorithmic Composition: Paradigms of Automated Music Generation.* SpringerWienNewYork, Wien, Austria, 1st edition.

Nishijima, M. and Watanabe, K. (1993). Interactive music composer based on neural networks. *Fujitsu Scientific Technical Journal*, pages 189–192.

ntu.edu.tw (2019). The MIDI File Format. `https://www.csie.ntu.edu.tw/~r92092/ref/midi/`. Accessed: 2019-05-15.

O'Brien, T. M. and Román, I. (2016). A Recurrent Neural Network for Musical Structure Processing and Expectation.

Papadopoulos, G. and Wiggins, G. (1999). AI Methods for Algorithmic Composition: A Survey, a Critical View and Future Prospects. *AISB Symposium on Musical Creativity*, pages 110–117.

Pascanu, R., Gülçehre, Ç., Cho, K., and Bengio, Y. (2013). How to Construct Deep Recurrent Neural Networks. *CoRR*, abs/1312.6026.

Prusinkiewicz, P. (1986). Score Generation with L-Systems. *Proceedings of the 1986 International Computer Music Conference*, pages 455–457.

Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. *CoRR*, abs/1511.06434.

Riedmiller, M. and Braun, H. (1993). A Direct Adaptive Method for Faster Backpropagation Learning: the RPROP Algorithm. *IEEE International Conference on Neural Networks*, pages 586–591.

Roberts, A., Engel, J., and Eck, D. (2017). Hierarchical Variational Autoencoders for Music. In *Workshop on Machine Learning for Creativity and Design, NIPS*.

Roberts, A., Engel, J., Raffel, C., Hawthorne, C., and Eck, D. (2018). A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music. *CoRR*, abs/1803.05428.

Ron, D., Singer, Y., and Tishby, N. (1996). The Power of Amnesia. *Machine Learning*, 25:117–149.

Sabathé, R., Coutinho, E., and Schuller, B. (2017). Deep recurrent music writer: Memory-enhanced variational autoencoder-based musical score composition and an objective measure. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 3467–3474.

Sarroff, A. M. and Casey, M. (2014). Musical audio synthesis using autoencoding neural nets. In *Joint 40th International Computer Music Conference (ICMC) and 11th Sound & Music Computing conference (SMC)*.

Shin, A., Crestel, L., Kato, H., Saito, K., Ohnishi, K., Yamaguchi, M., Nakawaki, M., Ushiku, Y., and Harada, T. (2017). Melody Generation for Pop Music via Word Representation of Musical Properties. *CoRR*, abs/1710.11549.

Sigtia, S., Benetos, E., Cherla, S., Weyde, T., Garcez, A., and Dixon, S. (2014). An RNN-based Music Language Model for Improving Automatic Music Transcription. In *International Society for Music Information Retrieval Conference (ISMIR)*.

Simon, I. and Oore, S. (2017). Performance RNN: Generating Music with Expressive Timing and Dynamics. `https://magenta.tensorflow.org/performance-rnn`.

Simon, I., Roberts, A., Raffel, C., Engel, J., Hawthorne, C., and Eck, D. (2018). Learning a Latent Space of Multitrack Measures. *CoRR*, abs/1806.00195.

Sohn, K., Lee, H., and Yan, X. (2015). Learning Structured Output Representation using Deep Conditional Generative Models. In *Advances in Neural Information Processing Systems*, pages 3483–3491.

Stange-Elbe, J. (2015). *Computer und Musik: Grundlagen, Technologien und Produktionsumgebungen der digitalen Musik*. De Gruyter Oldenbourg, 1st edition.

Sturm, B. L., Ben-Tal, O., Úna Monaghan, Collins, N., Herremans, D., Chew, E., Hadjeres, G., Deruty, E., and Pachet, F. (2018). Machine learning research that matters for music creation: A case study. *Journal of New Music Research*, 48(1):36–55.

Sturm, B. L., Santos, J. F., Ben-Tal, O., and Korshunova, I. (2016). Music transcription modelling and composition using deep learning. *CoRR*, abs/1604.08723.

Sun, F. (2015). DeepHear - Composing and harmonizing music with neural networks. `https://fephsun.github.io/2015/09/01/neural-music.html`.

Sun, Z., Liu, J., Zhang, Z., Chen, J., Huo, Z., Lee, C. H., and Zhang, X. (2016). Grammar Argumented LSTM Neural Networks with Note-Level Encoding for Music Composition. *CoRR*, abs/1611.05416.

Supper, M. (2001). A Few Remarks on Algorithmic Composition. *Computer Music Journal*, 25(1):48–53.

Teng, Y., Zhao, A., and Goudeseune, C. (2017). Generating Nontrivial Melodies for Music as a Service. *CoRR*, abs/1710.02280.

Tensorflow.org (2019). An End-to-end Open Source Machine Learning Platform. `https://www.tensorflow.org/`. Accessed: 2019-05-15.

Tikhonov, A. and Yamshchikov, I. P. (2017). Music Generation with Variational Recurrent Autoencoder Supported by History. *CoRR*, abs/1705.05458.

Todd, P. M. (1988). A Sequential Network Design for Musical Applications. *Proceedings of the 1988 Connectionist Models Summer School*, pages 76–84.

Todd, P. M. (1989). A Connectionist Approach to Algorithmic Composition. *Computer Music Journal*, 13(4):27–43.

Tsang, C. P. and Bellgard, M. I. (1992). Harmonizing music using a network of Boltzmann machines. *Proceedings of the Annual Conference of Artificial Neural Networks and their Applications*, pages 321–332.

Uria, B., Murray, I., and Larochelle, H. (2014). A Deep and Tractable Density Estimator. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages I–467–I–475. JMLR.org.

van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. W., and Kavukcuoglu, K. (2016a). WaveNet: A Generative Model for Raw Audio. *CoRR*, abs/1609.03499.

van den Oord, A., Kalchbrenner, N., and Kavukcuoglu, K. (2016b). Pixel Recurrent Neural Networks. *CoRR*, abs/1601.06759.

Verbeurgt, K., Fayer, M., and Michael, D. (2004). A Hybrid Neural-Markov Approach for Learning to Compose Music by Example. *Advances in Artificial Intelligence*, pages 480–484.

Vohra, R., Goel, K., and Sahoo, J. K. (2015). Modeling temporal dependencies in data using a DBN-LSTM. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*.

Walder, C. (2016). Modelling Symbolic Music: Beyond the Piano Roll. *CoRR*, abs/1606.01368.

Walder, C. J. and Kim, D. (2018). Neural Dynamic Programming for Musical Self Similarity. *CoRR*, abs/1802.03144.

Wang, S. and Manning, C. (2013). Fast Dropout Training. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 118–126, Atlanta, Georgia, USA. PMLR.

Wikipedia (2018a). Wikipedia (EN) "4'33"". `https://en.wikipedia.org/wiki/4%E2%80%B233%E2%80%B3`. Accessed: 2018-03-02.

Wikipedia (2018b). Wikipedia (EN) "Aleatoric music". `https://en.wikipedia.org/wiki/Aleatoric_music`. Accessed: 2018-03-02.

Wikipedia (2018c). Wikipedia (EN) "Klavierstücke (Stockhausen)". `https://en.wikipedia.org/wiki/Klavierst%C3%BCcke_(Stockhausen)`. Accessed: 2018-03-02.

Wikipedia (2018d). Wikipedia (EN) "Musikalisches Würfelspiel". `https://en.wikipedia.org/wiki/Musikalisches_W%C3%BCrfelspiel`. Accessed: 2018-03-02.

Wikipedia (2018e). Wikipedia (EN) "Twelve-tone technique". `https://en.wikipedia.org/wiki/Twelve-tone_technique`. Accessed: 2018-03-02.

Wu, J., Hu, C., Wang, Y., Hu, X., and Zhu, J. (2017). A Hierarchical Recurrent Neural Network for Symbolic Melody Generation. *CoRR*, abs/1712.05274.

Xenakis, I. (1992). *Formalized Music: Thought and Mathematics in Composition (Harmonologia Series no. 6)*. Pendragon Press, Stuyvesant, New York, USA, 2nd edition.

Yang, L.-C., Chou, S.-Y., and Yang, Y.-H. (2017). MidiNet: A convolutional Generative Adversarial Network for Symbolic-Domain Music Generation. *CoRR*.

Ycart, A. and Benetos, E. (2017). A study on LSTM networks for polyphonic music sequence modelling. In *18th International Society for Music Information Retrieval Conference*.

Yu, L., Zhang, W., Wang, J., and Yu, Y. (2016). SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient. *CoRR*.

Zilly, J. G., Srivastava, R. K., Koutník, J., and Schmidhuber, J. (2016). Recurrent Highway Networks. *CoRR*, abs/1607.03474.

APPENDICES

Literature Survey: Frameworks

A table over machine learning frameworks used in cited publications, when disclosed.

Publication (model)	Language	Frontend	Framework
Bayer et al., 2013	Python	-	Theano
Pascanu et al. (DT(S)-RNN, DOT(S)-RNN, sRNN), 2013	Python	-	Theano
Chung et al., 2014	Python	Pylearn2	Theano
Sarroff and Casey (DeepAutoController), 2014	Python	Pylearn2	Theano
Nayebi and Vitelli (GRUV), 2015	Python	Keras	Theano
Vohra et al. (LSTM-DBN), 2015	Python	-	Theano
Bretan et al., 2016	Python	-	Tensorflow
Choi et al. (char-RNN, word-RNN), 2016	Python	Keras	?
Colombo et al., 2016	Python	-	Theano
Huang and Wu, 2016	Python	-	Tensorflow
Kalingeri and Grandhe, 2016	Python	Keras	Tensorflow
Madjiheurem et al. (Chord2Vec), 2016	Python	-	Tensorflow
O'Brien and Román (MusicNet), 2016	Python	-	Tensorflow
Sun et al., 2016	Python	Keras	Tensorflow
Walder, 2016	Python	-	Tensorflow
Colombo et al. (DAC), 2017	Python	-	Theano
Johnson (LSTM-NADE, TP-LSTM-NADE, BALSTM), 2017	Python	-	Theano
Mehri et al. (SampleRNN), 2017	Python	-	Theano
Engel et al., 2017	Python	-	Tensorflow
Hadjeres and Pachet (DeepBach), 2017	Python	Keras	Tensorflow
Hadjeres and Nielsen (Anticipation-RNN), 2017	Python	-	PyTorch
Hennig et al. (Classifying VAE, Classifying VAE+LSTM), 2017	Python	Keras	Tensorflow
Jaques et al. (Sequence Tutor), 2017	Python	-	Tensorflow
Roberts et al. (MusicVAE), 2017	Python	-	Tensorflow
Roberts et al. (MusicVAE), 2018	Python	-	Tensorflow
Sabathé et al. (DRAW), 2017	Python	-	Tensorflow
Simon and Oore (PerformanceRNN), 2017	Python	-	Tensorflow
Teng et al., 2017	Python	-	Tensorflow
Ycart and Benetos, 2017	Python	-	Tensorflow
Yang et al. (MidiNet), 2017	Python	-	Tensorflow
Colombo and Gerstner (BachProp), 2018	Python	Keras	?
Simon et al. (MusicVAE), 2018	Python	-	Tensorflow
Walder and Kim (MotifNet), 2018	Python	-	PyTorch

Literature Survey: Preprocessing Frameworks

A table over preprocessing frameworks used to modify input representation to internal data representation used in cited publications, when applicable.

Publication (model)	Format	Language	Framework
Brunner et al. (JamBot), 2017	MIDI	Python	pretty-midi
Colombo et al. (DAC), 2017	MIDI	Python	music21
Hadjeres et al. (GLSR-VAE), 2017	MIDI	Python	music21
Lee et al. (SeqGAN), 2017	MIDI	Python	music21
Shin et al., 2017	MIDI	Python	pretty-midi
Shin et al., 2017	MIDI	Python	mido
Simon et al. (MusicVAE), 2018	MIDI	Python	pretty-midi

Literature Survey: Datasets

A table over datasets used in cited publications, where disclosed.

Dataset	Link	Genre	Format
The Session	`http://www.thesession.org`	Folk	ABC
Nottingham Database	`http://www.cs.nott.ac.uk/~ef/music/database.htm`	Folk	ABC
Nottingham Database	`http://ifdo.ca/~seymour/nottingham/nottingham.html`	Folk	ABC
Nottingham Database	`http://www-etud.iro.umontreal.ca/~boulanni/icml2012`	Folk	MIDI
Nottingham Database	`http://www-etud.iro.umontreal.ca/~boulanni/icml2012`	Folk	Piano roll
Nottingham Database	`http://users.cecs.anu.edu.au/~u1018264/data.html`	Folk	MIDI-inspired
MuseData	`http://www.musedata.org`	Classical	Humdrum / MIDI
MuseData	`http://www-etud.iro.umontreal.ca/~boulanni/icml2012`	Classical	MIDI
MuseData	`http://www-etud.iro.umontreal.ca/~boulanni/icml2012`	Classical	Piano roll
MuseData	`http://users.cecs.anu.edu.au/~u1018264/data.html`	Classical	MIDI-inspired
Piano-midi.de	`http://www.piano-midi.de/`	Classical	MIDI
Piano-midi.de	`http://www-etud.iro.umontreal.ca/~boulanni/icml2012`	Classical	MIDI
Piano-midi.de	`http://www-etud.iro.umontreal.ca/~boulanni/icml2012`	Classical	Piano roll
Piano-midi.de	`http://users.cecs.anu.edu.au/~u1018264/data.html`	Classical	MIDI-inspired
JSB Chorales	`http://www-etud.iro.umontreal.ca/~boulanni/icml2012`	Classical	MIDI
JSB Chorales	`http://www-etud.iro.umontreal.ca/~boulanni/icml2012`	Classical	Piano roll
JSB Chorales	`http://users.cecs.anu.edu.au/~u1018264/data.html`	Classical	MIDI-inspired
Bach10	`http://music.cs.northwestern.edu/data/Bach10.html`	Classical	Audio

Wikifonia	`http://www.synthzone.com/files/Wikifonia/Wikifonia.zip`	Folk, pop, rock, classical	MusicXML
midi_man	`https://redd.it/3ajwe4`	Pop, video game, classical	MIDI
Henrik Norbeck's ABC Tunes	`http://www.norbeck.nu/abc/`	Folk	ABC
MagnaTag-ATune	`http://mirg.city.ac.uk/codeapps/the-magnatagatune-dataset`	Mixed	Audio
Lakh	`https://colinraffel.com/projects/lmd/`	Mixed	MIDI
Million Song Dataset	`https://labrosa.ee.columbia.edu/millionsong/`	Mixed	MIDI
NSynth	`https://magenta.tensorflow.org/datasets/nsynth`	Instrument timbres	Audio
Yamaha e-Piano Competition datasets	`http://www.piano-e-competition.com/midiinstructions.asp`	classical piano	MIDI
Dave's J.S. Bach	`http://www.jsbach.net`	Bach	MIDI
John Sankey Bach	`http://www.jsbach.net`	Bach	MIDI
String quartets	`http://www.stringquartets.org`	Mozart, Haydn	MIDI

Literature Survey: Results

Tables over results over the most well-known benchmark datasets.

Nottingham Dataset

Publication (model)	Test loss	Test acc. (%)
Boulanger-Lewandowski et al. (RANDOM), 2012	-61.00	4.53
Boulanger-Lewandowski et al. (RBM), 2012	-5.25	5.81
Boulanger-Lewandowski et al. (NADE), 2012	-5.48	22.67
Boulanger-Lewandowski et al. (MLP), 2012	-4.38	63.46
Boulanger-Lewandowski et al. (RNN), 2012	-4.46	62.93
Boulanger-Lewandowski et al. (RNN (HF)), 2012	-3.89	66.64
Boulanger-Lewandowski et al. (RTRBM), 2012	-2.62	75.01
Boulanger-Lewandowski et al. (RNN-RBM), 2012	-2.39	75.40
Boulanger-Lewandowski et al. (RNN-NADE), 2012	-2.91	64.95
Boulanger-Lewandowski et al. (RNN-NADE (HF)), 2012	-2.31	71.50
Bengio et al. (RNN (SGD)), 2012	-3.75	66.90
Bengio et al. (RNN (SGD+C)), 2012	-3.67	67.47
Bengio et al. (RNN (SGD+CL)), 2012	-3.57	67.97
Bengio et al. (RNN (SGD+CLR)), 2012	-3.55	70.20
Bengio et al. (RNN (SGD+CRM)), 2012	-3.43	68.47
Bengio et al. (RNN (HF)), 2012	-3.76	66.71
Bengio et al. (RNN-NADE (SGD)), 2012	-2.91	64.95
Bengio et al. (RNN-NADE (SGD+CR), 2012	-2.51	69.80
Bengio et al. (RNN-NADE (SGD+CRM)), 2012	-2.49	69.54
Pascanu et al. (RNN), 2013	-3.225	N/A
Pascanu et al. (DT(S)-RNN), 2013	-3.206	N/A
Pascanu et al. (DOT(S)-RNN), 2013	-3.215	N/A
Pascanu et al. (sRNN), 2013	-3.258	N/A
Pascanu et al. (DOT(S)-RNN (maxout + Lp)), 2013	-2.95	N/A
Chung et al. (RNN(tanh)), 2014	-3.13	N/A
Chung et al. (GRU), 2014	-3.23	N/A
Chung et al. (LSTM), 2014	-3.20	N/A
Bayer et al. (RNN (FD)), 2013	-3.09	N/A
Goel et al. (RNN-DBN), 2014	-2.54	N/A
Sigtia et al. (RNN (SGD)), 2014	N/A	67.89
Sigtia et al. (RNN (HF)), 2014	N/A	69.61
Sigtia et al. (RNN-NADE (SGD)), 2014	N/A	68.89
Sigtia et al. (RNN-NADE (HF)), 2014	N/A	70.61
Bayer and Osendorfer (STORN), 2015	-2.85	N/A
Gan et al. (TSBN), 2015	-3.67	N/A

Vohra et al. (LSTM-DBN), 2015	-1.32	N/A
Walder (LSTM), 2016	-2.06	N/A
Walder (LSTM (augmented)), 2016	-1.66	N/A
Walder (LSTM (augmented + pooled)), 2016	-1.57	N/A
Johnson (LSTM-NADE), 2017	-2.02	N/A
Johnson (TP-LSTM-NADE), 2017	-1.61	N/A
Johnson (BALSTM), 2017	-1.55	N/A

Musedata Dataset

Publication (model)	Test loss	Test acc. (%)
Boulanger-Lewandowski et al. (RANDOM), 2012	-61.00	3.74
Boulanger-Lewandowski et al. (RBM), 2012	-9.56	8.19
Boulanger-Lewandowski et al. (NADE), 2012	-10.06	7.65
Boulanger-Lewandowski et al. (MLP), 2012	-7.94	25.68
Boulanger-Lewandowski et al. (RNN), 2012	-8.13	23.25
Boulanger-Lewandowski et al. (RNN (HF)), 2012	-7.19	30.49
Boulanger-Lewandowski et al. (RTRBM), 2012	-6.35	30.85
Boulanger-Lewandowski et al. (RNN-RBM), 2012	-6.01	34.02
Boulanger-Lewandowski et al. (RNN-NADE), 2012	-6.74	24.91
Boulanger-Lewandowski et al. (RNN-NADE (HF)), 2012	-5.60	32.60
Bengio et al. (RNN (SGD)), 2012	-7.20	27.97
Bengio et al. (RNN (SGD+C)), 2012	-7.04	30.53
Bengio et al. (RNN (SGD+CL)), 2012	-6.99	31.53
Bengio et al. (RNN (SGD+CLR)), 2012	-7.34	29.06
Bengio et al. (RNN (SGD+CRM)), 2012	-7.24	29.13
Bengio et al. (RNN (HF)), 2012	-7.12	29.77
Bengio et al. (RNN-NADE (SGD)), 2012	-6.74	24.91
Bengio et al. (RNN-NADE (SGD+CR), 2012	-6.37	26.60
Bengio et al. (RNN-NADE (SGD+CRM)), 2012	-6.19	29.62
Pascanu et al. (RNN), 2013	-6.990	N/A
Pascanu et al. (DT(S)-RNN), 2013	-6.988	N/A
Pascanu et al. (DOT(S)-RNN), 2013	-6.973	N/A
Pascanu et al. (sRNN), 2013	-6.954	N/A
Pascanu et al. (DOT(S)-RNN (maxout + Lp)), 2013	-6.59	N/A
Chung et al. (RNN(tanh)), 2014	-6.23	N/A
Chung et al. (GRU), 2014	-5.99	N/A
Chung et al. (LSTM), 2014	-6.23	N/A
Bayer et al. (RNN (FD)), 2013	-6.75	N/A
Goel et al. (RNN-DBN), 2014	-6.28	N/A

Bayer and Osendorfer (STORN), 2015	-6.16	N/A
Gan et al. (TSBN), 2015	-6.81	N/A
Lyu et al. (LSTM-RTRBM), 2015	-5.54	N/A
Vohra et al. (LSTM-DBN), 2015	-3.91	N/A
Walder (LSTM), 2016	-5.16	N/A
Walder (LSTM (augmented)), 2016	-4.46	N/A
Walder (LSTM (augmented + pooled)), 2016	-4.41	N/A
Johnson (LSTM-NADE), 2017	-5.02	N/A
Johnson (TP-LSTM-NADE), 2017	-4.32	N/A
Johnson (BALSTM), 2017	-3.90	N/A

Pianomidi Dataset

Publication (model)	Test loss	Test acc. (%)
Boulanger-Lewandowski et al. (RANDOM), 2012	-61.00	3.35
Boulanger-Lewandowski et al. (RBM), 2012	-10.17	5.63
Boulanger-Lewandowski et al. (NADE), 2012	-10.28	5.82
Boulanger-Lewandowski et al. (MLP), 2012	-8.13	20.29
Boulanger-Lewandowski et al. (RNN), 2012	-8.37	19.33
Boulanger-Lewandowski et al. (RNN (HF)), 2012	-7.66	23.34
Boulanger-Lewandowski et al. (RTRBM), 2012	-7.36	22.99
Boulanger-Lewandowski et al. (RNN-RBM), 2012	-7.09	28.92
Boulanger-Lewandowski et al. (RNN-NADE), 2012	-7.48	20.69
Boulanger-Lewandowski et al. (RNN-NADE (HF)), 2012	-7.05	23.42
Bengio et al. (RNN (SGD)), 2012	-7.86	22.84
Bengio et al. (RNN (SGD+C)), 2012	-7.59	22.98
Bengio et al. (RNN (SGD+CL)), 2012	-7.57	22.97
Bengio et al. (RNN (SGD+CLR)), 2012	-7.80	24.22
Bengio et al. (RNN (SGD+CRM)), 2012	-7.73	23.71
Bengio et al. (RNN (HF)), 2012	-7.58	22.93
Bengio et al. (RNN-NADE (SGD)), 2012	-7.48	20.69
Bengio et al. (RNN-NADE (SGD+CR), 2012	-7.34	21.22
Bengio et al. (RNN-NADE (SGD+CRM)), 2012	-7.34	22.12
Bayer et al. (RNN (FD)), 2013	-7.39	N/A
Chung et al. (RNN(tanh)), 2014	-9.03	N/A
Chung et al. (GRU), 2014	-8.82	N/A
Chung et al. (LSTM), 2014	-9.03	N/A
Goel et al. (RNN-DBN), 2014	-7.15	N/A
Bayer and Osendorfer (STORN), 2015	-7.13	N/A
Gan et al. (TSBN), 2015	-7.98	N/A

Vohra et al. (LSTM-DBN), 2015	-4.63	N/A
Walder (LSTM), 2016	-6.67	N/A
Walder (LSTM (augmented)), 2016	-5.43	N/A
Walder (LSTM (augmented + pooled)), 2016	-4.94	N/A
Johnson (LSTM-NADE), 2017	-7.36	N/A
Johnson (TP-LSTM-NADE), 2017	-5.44	N/A
Johnson (BALSTM), 2017	-4.90	N/A
Hennig et al. (VAE), 2017	-7.51	N/A
Hennig et al. (Classifying VAE), 2017	-7.05	N/A
Hennig et al. (VAE+LSTM), 2017	-7.49	N/A
Hennig et al. (Classifying VAE+LSTM), 2017	-7.11	N/A

JSBChorales Dataset

Publication (model)	Test loss	Test acc. (%)
Boulanger-Lewandowski et al. (RANDOM), 2012	-61.00	4.42
Boulanger-Lewandowski et al. (RBM), 2012	-7.43	4.47
Boulanger-Lewandowski et al. (NADE), 2012	-7.19	17.88
Boulanger-Lewandowski et al. (MLP), 2012	-8.70	30.41
Boulanger-Lewandowski et al. (RNN), 2012	-8.71	28.46
Boulanger-Lewandowski et al. (RNN (HF)), 2012	-8.58	29.41
Boulanger-Lewandowski et al. (RTRBM), 2012	-6.35	30.17
Boulanger-Lewandowski et al. (RNN-RBM), 2012	-6.27	33.12
Boulanger-Lewandowski et al. (RNN-NADE), 2012	-5.83	32.11
Boulanger-Lewandowski et al. (RNN-NADE (HF)), 2012	-5.56	32.50
Bengio et al. (RNN (SGD)), 2012	-8.65	29.97
Bengio et al. (RNN (SGD+C)), 2012	-8.65	29.98
Bengio et al. (RNN (SGD+CL)), 2012	-8.63	29.98
Bengio et al. (RNN (SGD+CLR)), 2012	-9.47	29.98
Bengio et al. (RNN (SGD+CRM)), 2012	-8.81	29.52
Bengio et al. (RNN (HF)), 2012	-8.58	29.41
Bengio et al. (RNN-NADE (SGD)), 2012	-5.83	32.11
Bengio et al. (RNN-NADE (SGD+CR), 2012	-5.33	34.52
Bengio et al. (RNN-NADE (SGD+CRM)), 2012	-5.19	35.08
Bayer et al. (RNN (FD)), 2013	-8.01	N/A
Pascanu et al. (RNN), 2013	-8.338	N/A
Pascanu et al. (DT(S)-RNN), 2013	-8.278	N/A
Pascanu et al. (DOT(S)-RNN), 2013	-8.437	N/A
Pascanu et al. (sRNN), 2013	-8.367	N/A
Pascanu et al. (DOT(S)-RNN (maxout + Lp)), 2013	-7.92	N/A

Chung et al. (RNN(tanh)), 2014	-9.10	N/A
Chung et al. (GRU), 2014	-8.54	N/A
Chung et al. (LSTM), 2014	-8.67	N/A
Goel et al. (RNN-DBN), 2014	-5.68	N/A
Liu and Ramakrishnan (BPTT), 2014	N/A	21.03
Liu and Ramakrishnan (RProp), 2014	N/A	31.91
Bayer and Osendorfer (STORN), 2015	-6.91	N/A
Gan et al. (TSBN), 2015	-7.48	N/A
Lyu et al. (LSTM-RTRBM), 2015	-4.72	N/A
Vohra et al. (LSTM-DBN), 2015	-3.47	N/A
Walder (LSTM), 2016	-5.01	N/A
Walder (LSTM (augmented)), 2016	-4.34	N/A
Walder (LSTM (augmented + pooled)), 2016	-4.45	N/A
Johnson (LSTM-NADE), 2017	-6.00	N/A
Johnson (TP-LSTM-NADE), 2017	-5.88	N/A
Johnson (BALSTM), 2017	-5.05	N/A
Hennig et al. (VAE), 2017	-6.87	N/A
Hennig et al. (Classifying VAE), 2017	-6.66	N/A
Hennig et al. (VAE+LSTM), 2017	-6.83	N/A
Hennig et al. (Classifying VAE+LSTM), 2017	-6.73	N/A
Huang et al. (RNN-NADE), 2017	-5.03	N/A
Huang et al. (CocoNet (chronological ordering)), 2017	-7.79	N/A
Huang et al. (CocoNet (random ordering)), 2017	-5.03	N/A

Literature survey: Sources

Links to source code from cited publications where available.

Publication (model)	Link
Bickerman et al. (RBM-Provisor), 2010	https://sourceforge.net/projects/rbm-provisor/
Boulanger-Lewandowski et al. (RNN-RBM), 2012 (reimpl.)	http://deeplearning.net/tutorial/rnnrbm.html
Chung et al., 2014	https://github.com/jych/librnn.git
Sarroff and Casey (DeepAutoController), 2014	https://github.com/woodshop/deepAutoController
Gan et al. (TSBN), 2015	https://github.com/zhegan27/TSBN_code_NIPS2015
Sun (DeepHear), 2015	https://github.com/fephsun/neuralnetmusic
Choi et al. (char-RNN, word-RNN), 2016	https://github.com/keunwoochoi/lstm_real_book
Choi et al. (char-RNN, word-RNN), 2016	https://github.com/keunwoochoi/LSTMetallica
Jaques et al. (RL Tuner), 2016	https://github.com/natashamjaques/magenta/tree/rl-tuner
Mogren (C-RNN-GAN), 2016	https://github.com/olofmogren/c-rnn-gan
O'Brien and Román (MusicNet), 2016	https://cm-gitlab.stanford.edu/tsob/musicNet
Sturm et al. (char-rnn, folk-rnn), 2016	https://github.com/IraKorshunova/folk-rnn
Yu et al. (SeqGAN), 2016	https://github.com/LantaoYu/SeqGAN
Agarwala et al., 2017	https://github.com/yinoue93/CS224N_proj
Brunner et al. (JamBot), 2017	https://github.com/brunnergino/JamBot
Chen et al. (FusionGAN), 2017	https://github.com/aquastar/fusion_gan
Johnson (LSTM-NADE, TP-LSTM-NADE, BALSTM), 2017	https://github.com/hexahedria/biaxial-rnn-music-composition
Mehri et al. (SampleRNN), 2017	https://github.com/soroushmehr/sampleRNN_ICLR2017
Dong et al. (MuseGAN), 2017	https://github.com/salu133445/musegan
Guimaraes et al. (ORGAN), 2017	https://github.com/gablg1/ORGAN
Hadjeres and Pachet (DeepBach), 2017	https://github.com/Ghadjeres/DeepBach

Hadjeres and Nielsen (Anticipation-RNN), 2017	`https://github.com/Ghadjeres/Anticipation-RNN`
Hennig et al. (Classifying VAE, Classifying VAE+LSTM), 2017	`https://github.com/mobeets/classifying-vae-lstm`
Huang et al. (CocoNet), 2017	`https://github.com/czhuang/coconet`
Jaques et al. (Sequence Tutor), 2017	`https://github.com/tensorflow/magenta/tree/master/magenta/models/rl_tuner`
Liang et al. (BachBot), 2017	`https://github.com/feynmanliang/bachbot`
Roberts et al. (MusicVAE), 2017	`https://github.com/tensorflow/magenta/tree/master/magenta/models/music_vae`
Roberts et al. (MusicVAE), 2018	`https://github.com/tensorflow/magenta/tree/master/magenta/models/music_vae`
Shin et al., 2017	`https://github.com/mil-tokyo/NeuralMelody`
Simon and Oore (PerformanceRNN), 2017	`https://github.com/tensorflow/magenta/tree/master/magenta/models/performance_rnn`
Ycart and Benetos, 2017	`https://code.soundsoftware.ac.uk/projects/ismir17-code`
Yang et al. (MidiNet), 2017	`https://github.com/RichardYang40148/MidiNet`
Colombo and Gerstner (BachProp), 2018	`https://github.com/FlorianColombo/BachProp`
Mao et al. (DeepJ), 2018	`https://github.com/calclavia/DeepJ/tree/icsc`
Walder and Kim (MotifNet), 2018	`https://bitbucket.org/cwalder/motif`

Literature survey: Samples

Links to listening samples from cited application where available.

Publication (model)	Link
Eck and Schmidhuber, 2002	http://people.idsia.ch/~juergen/blues/index.html
Boulanger-Lewandowski et al. (RNN-RBM, RNN-NADE), 2012	http://www-etud.iro.umontreal.ca/~boulanni/icml2012
Sarroff and Casey (DeepAutoController), 2014	https://andysarroff.com/projects/autoencoding-synthesizers/
Fabius and van Amersfoort (VRAE), 2015	http://youtu.be/cu1_uJ9qkHA
Gan et al. (TSBN), 2015	https://drive.google.com/drive/u/0/folders/0B1HR6m3IZSO_SWt0aS1oYmlneDQ
Lyu et al. (LSTM-RTRBM), 2015	https://bitbucket.org/huashiyiqike/music-samples
Nayebi and Vitelli (GRUV), 2015	https://www.youtube.com/watch?v=0VTI1BBLydE
Sun (DeepHear), 2015	https://fephsun.github.io/2015/09/01/neural-music.html
Bretan et al., 2016	https://youtu.be/BbyvbO2F7ug
Choi et al. (char-RNN, word-RNN), 2016	https://soundcloud.com/kchoi-research/sets/lstm-realbook-1-5
Choi et al. (char-RNN, word-RNN), 2016	https://soundcloud.com/kchoi-research/sets/lstmetallica-drums
Choi et al. (char-RNN, word-RNN), 2016	https://soundcloud.com/kchoi-research/00-24-100-bonus-for-score
Chu et al., 2016	http://www.cs.toronto.edu/songfrompi/
Jaques et al. (RL Tuner), 2016	http://goo.gl/XIYt9m/
Lattner et al. (C-RBM), 2016	https://soundcloud.com/pmgrbm
Mogren (C-RNN-GAN), 2016	http://mogren.one/publications/2016/c-rnn-gan/
Sturm et al. (char-rnn, folk-rnn), 2016	https://github.com/IraKorshunova/folk-rnn
Walder, 2016	http://users.cecs.anu.edu.au/~u1018264/beyondthepianoroll.html
van den Oord et al. (WaveNet), 2016a	https://deepmind.com/blog/wavenet-generative-model-raw-audio/
Agarwala et al., 2017	https://yinoue93.github.io/CS224N.html
Brunner et al. (JamBot), 2017	https://www.youtube.com/channel/UCQbE9vfbYycK4DZpHoZKcSw

Chen et al. (FusionGAN), 2017	`http://people.cs.vt.edu/czq/publication/fusiongan/`
Colombo et al. (DAC), 2017	`https://goo.gl/CHhoAu`
Johnson (LSTM-NADE, TP-LSTM-NADE, BALSTM), 2017	`https://www.cs.hmc.edu/~ddjohnson/tied-parallel/`
Mehri et al. (SampleRNN), 2017	`https://soundcloud.com/samplernn/sets`
De Prisco et al., 2017	`http://goo.gl/FWn2EX`
Dong et al. (MuseGAN), 2017	`https://salu133445.github.io/musegan/`
Engel et al., 2017	`https://magenta.tensorflow.org/nsynth`
Hadjeres and Pachet (DeepBach), 2017	`https://sites.google.com/site/deepbachexamples/`
Hennig et al. (Classifying VAE, Classifying VAE+LSTM), 2017	`https://mobeets.github.io/classifying-vae-lstm/`
Huang et al. (CocoNet), 2017	`https://coconets.github.io/`
Jaques et al. (Sequence Tutor), 2017	`http://goo.gl/XIYt9m/`
Lee et al. (SeqGAN), 2017	`https://soundcloud.com/sang-gil-lee-474904648/tracks`
Liang et al. (BachBot), 2017	`http://bachbot.com`
Malik and Ek (StyleNet), 2017	`http://imanmalik.com/cs/2017/06/05/neural-style.html`
Roberts et al. (MusicVAE), 2017	`https://www.youtube.com/playlist?list=PLBUMAYA6kvGU8Cgqh709o5SUvo-zHGTxr`
Roberts et al. (MusicVAE), 2018	`http://g.co/magenta/musicvae-samples`
Sabathé et al. (DRAW), 2017	`http://www.openaudio.eu/`
Shin et al., 2017	`https://soundcloud.com/iclr2018eval`
Simon and Oore (PerformanceRNN), 2017	`https://magenta.tensorflow.org/performance-rnn`
Teng et al., 2017	`https://composing.ai/pieces`
Tikhonov and Yamshchikov (VRASH), 2017	`https://soundcloud.com/creaited-labs/`
Wu et al. (HRNN), 2017	`https://www.dropbox.com/s/vnd6hoq9olrpb5g/SM.zip?dl=0`
Yang et al. (MidiNet), 2017	`https://soundcloud.com/vgtsv6jf5fwq/sets`
Colombo and Gerstner (BachProp), 2018	`https://goo.gl/Xyx7WV`
Eppe et al., 2018	`http://www.publications.eppe.eu/data/Giuliani_Op74_No15_Andantino_grazioso_merged`
Eppe et al., 2018	`http://www.publications.eppe.eu/data/The_Beatles_Ob-La-Di_Ob-La-Da_merged.wav`

Eppe et al., 2018	`http://www.publications.eppe.eu/data/Bob_Dylan_Positively_4th_Street_merged.wav`
Koh et al. (C-RVAE), 2018	`https://soundcloud.com/user-431911640/sets`
Mao et al. (DeepJ), 2018	`https://github.com/calclavia/DeepJ/tree/icsc/archives/v1`
Simon et al. (MusicVAE), 2018	`https://goo.gl/s2N7dV`

Method: Datasets

Information and links to source locations of used datasets.

Id	Name	Source
ESSEN	EsAC - Essen Associative Code and Folksong Database	`http://kern.ccarh.org/browse?l=essen`
SESSION	The Session	`https://github.com/IraKorshunova/folk-rnn/tree/master/data`
NOTTINGHAM	Nottingham Database	`http://www-etud.iro.umontreal.ca/~boulanni/icml2012`
PIANOMIDI	Piano-midi.de	`http://www.piano-midi.de/`
MUSEDATA	MuseData Musical Data	`http://www-etud.iro.umontreal.ca/~boulanni/icml2012`
MAHLER	(Custom dataset)	`http://gustavmahler.com/midi.html` `http://midi-orchestra.xii.jp/ta/mahler.htm` `https://www.youtube.com/watch?v=K-3NFGlDPKk`

Results: Samples

Overview of listening samples from experiments available at `http://www.mahlernet.se`. The bars column refers to 1-indexed bars in the source file, where applicable, with START signifying the use of a vector with only the special START token. The dataset column indicates which dataset of ESSEN (ES), SESSION (SE), NOTTINGHAM (NO), PIANOMIDI (PI), MUSEDATA (MU) and MAHLER (MA) the model that generated the sample was trained on. It is also indicated if the sample was generated with a regular RNN decoder (R) an RNN decoder with scheduled sampling (S) or a BALSTM (B) and if it was a model that used context (C) or not. More details on the setups can be found in section 5.

Experiment 1 with Pitch Only

Name	Type	File	Bars	Dataset
1-1	10-step interpolation	sessiontune6	13, 17	SE(R)
1-2	10-step interpolation	sessiontune14425	1, 4	SE(R)
1-3	10-step interpolation	sessiontune6	13, 17	SE(S)
1-4	10-step interpolation	sessiontune14425	1, 4	SE(S)
1-5	10-step interpolation	sessiontune6	13, 17	SE(B)
1-6	10-step interpolation	sessiontune14425	1, 4	SE(B)
1-7	10-step interpolation	essen_europa_sverige_sverig03	2, 7	ES(R)
1-8	10-step interpolation	essen_europa_sverige_sverig03	2, 7	ES(S)
1-9	10-step interpolation	essen_europa_sverige_sverig03	2, 7	ES(B)
1-10	1-bar random sample	-	-	SE(R)
1-11	1-bar random sample	-	-	SE(S)
1-12	1-bar random sample	-	-	SE(B)
1-13	1-bar random sample	-	-	ES(R)
1-14	1-bar random sample	-	-	ES(S)
1-15	1-bar random sample	-	-	ES(B)
1-16	10-bar seeded sample	sessiontune0	START, 1	SE(R)
1-17	10-bar seeded sample	sessiontune0	START, 1	SE(S)
1-18	10-bar seeded sample	sessiontune0	START, 1	SE(B)
1-19	10-bar seeded sample	essen_europa_sverige_sverig06	START, 1	ES(R)
1-20	10-bar seeded sample	essen_europa_sverige_sverig06	START, 1	ES(S)
1-21	10-bar seeded sample	essen_europa_sverige_sverig06	START, 1	ES(B)
1-22	10-bar seeded sample	sessiontune0	START, 1	SE(RC)
1-23	10-bar seeded sample	sessiontune0	START, 1	SE(SC)
1-24	10-bar seeded sample	sessiontune0	START, 1	SE(BC)
1-25	10-bar seeded sample	essen_europa_sverige_sverig06	START, 1	ES(RC)
1-26	10-bar seeded sample	essen_europa_sverige_sverig06	START, 1	ES(SC)
1-27	10-bar seeded sample	essen_europa_sverige_sverig06	START, 1	ES(BC)

Experiment 2 with All Properties

Name	Type	File	Bars	Dataset
2-1	10-step interpolation	mz_333_1	137, 204	PI(R)
2-2	5-step interpolation	pathetique_1	106, 111	PI(R)
2-3	1-bar random sample	-	-	PI(R)
2-4	10-bar random sample	-	-	PI(R)
2-5	10-step interpolation	mz_333_1	137, 204	PI(S)
2-6	5-step interpolation	pathetique_1	106, 111	PI(S)
2-7	1-bar random sample	-	-	PI(S)
2-8	10-step interpolation	mz_333_1	137, 204	PI(B)
2-9	5-step interpolation	pathetique_1	106, 111	PI(B)
2-10	1-bar random sample	-	-	PI(B)
2-11	10-bar random sample	-	-	PI(R)
2-12	10-step interpolation	mozart.conc.k466_midi1_01	48, 51	MU(R)
2-13	1-bar random sample	-	-	MU(R)
2-14	10-step interpolation	mozart.conc.k466_midi1_01	48, 51	MU(S)
2-15	1-bar random sample	-	-	MU(S)
2-16	10-step interpolation	mozart.conc.k466_midi1_01	48, 51	MU(B)
2-17	1-bar random sample	-	-	MU(B)
2-18	10-bar seeded sample	elise	1, 2	PI(RC)
2-19	10-bar seeded sample	mz_333_1	START, 1	PI(RC)
2-20	10-bar seeded sample	mz_333_1	45, 46	PI(RC)
2-21	10-bar seeded sample	mz_333_1	122, 123	PI(RC)
2-22	10-bar seeded sample	mz_333_1	140, 141	PI(RC)
2-23	10-bar seeded sample	pathetique_3	1, 2	PI(RC)
2-24	10-bar seeded sample	pathetique_3	58, 59	PI(RC)
2-25	10-bar seeded sample	chpn_op10_e01	1, 2	PI(RC)
2-26	10-bar random sample	-	START	PI(RC)
2-27	10-bar random sample	-	START	PI(RC)
2-28	10-bar random sample	-	START	PI(RC)
2-29	10-bar random sample	-	START	PI(RC)
2-30	10-bar random sample	-	START	PI(RC)
2-31	100-bar random sample	-	START	PI(RC)
2-32	10-bar seeded sample	chpn_op10_e01	1, 2	PI(SC)
2-33	10-bar seeded sample	elise	1, 2	PI(SC)
2-34	10-bar seeded sample	mz_333_1	45, 46	PI(SC)
2-35	10-bar seeded sample	mz_333_1	122, 123	PI(SC)
2-36	10-bar seeded sample	mz_333_1	140, 141	PI(SC)
2-37	10-bar seeded sample	pathetique_3	1, 2	PI(SC)
2-38	10-bar seeded sample	pathetique_3	58, 59	PI(SC)

2-39	10-bar random sample	-	START	PI(SC)
2-40	10-bar random sample	-	START	PI(SC)
2-41	10-bar random sample	-	START	PI(SC)
2-42	10-bar random sample	-	START	PI(SC)
2-43	100-bar random sample	-	START	PI(SC)
2-44	10-bar seeded sample	beet.conc.violin_midi1_01	2, 3	MU(RC)
2-45	10-bar seeded sample	mozart.conc.k466_midi1_03	20, 21	MU(RC)
2-46	10-bar seeded sample	mozart.conc.k466_midi1_03	30, 31	MU(RC)
2-47	10-bar seeded sample	mozart.conc.k466_midi1_03	52, 53	MU(RC)
2-48	10-bar seeded sample	mozart.conc.k466_midi1_01	44, 45	MU(RC)
2-49	10-bar seeded sample	mozart.conc.k622_midi1_01	1, 2	MU(RC)
2-50	10-bar seeded sample	mozart.sym.k550_midi1_03	1, 2	MU(RC)
2-51	10-bar random sample	-	START	MU(RC)
2-52	10-bar random sample	-	START	MU(RC)
2-53	10-bar random sample	-	START	MU(RC)
2-54	10-bar random sample	-	START	MU(RC)
2-55	10-bar random sample	-	START	MU(RC)
2-56	10-bar random sample	-	START	MU(RC)
2-57	10-bar random sample	-	START	MU(RC)
2-58	100-bar random sample	-	START	MU(RC)
2-59	100-bar random sample	-	START	MU(RC)
2-60	10-bar seeded sample	beet.conc.violin_midi1_01	1, 2	MU(SC)
2-61	10-bar seeded sample	beet.conc.violin_midi1_01	2, 3	MU(SC)
2-62	10-bar seeded sample	mozart.conc.k466_midi1_03	30, 31	MU(SC)
2-63	10-bar seeded sample	mozart.conc.k466_midi1_03	52, 53	MU(SC)
2-64	10-bar random sample	-	START	MU(SC)
2-65	10-bar random sample	-	START	MU(SC)
2-66	10-bar random sample	-	START	MU(SC)
2-67	100-bar random sample	-	START	MU(SC)

Experiment 3 with All Properties

Name	Type	File	Bars	Dataset
3-1	10-step interpolation	Mahlsy54	10, 11	MA(R)
3-2	10-step interpolation	Mahler61	14, 470	MA(R)
3-3	10-step interpolation	Mahler61	60, 61	MA(R)
3-4	10-step interpolation	3rd-movement-Feierlich-und-gemessen-ohne-zu-schleppen.midi	8, 22	MA(R)

3-5	10-step interpolation	concatenation of MAHL64O1, MAHL64O2 and MAHL64O3	689, 691	MA(R)
3-6	10-step interpolation	Mahler61	2, 6	MA(S)
3-7	10-step interpolation	Mahler61	581, 583	MA(S)
3-8	10-step interpolation	Mahler63	79, 80	MA(S)
3-9	10-step interpolation	concatenation of MAHL64O1, MAHL64O2 and MAHL64O3	119, 121	MA(S)
3-10	10-bar seeded sample	Mahler61	58, 59	MA(RC)
3-11	10-bar seeded sample	Mahler61	59, 60	MA(RC)
3-12	10-bar seeded sample	Mahler61	62, 63	MA(RC)
3-13	10-bar seeded sample	Mahler61	134, 135	MA(RC)
3-14	10-bar seeded sample	Mahler61	264, 265	MA(RC)
3-15	10-bar seeded sample	Mahler61	264, 265	MA(RC)
3-16	10-bar seeded sample	Mahler61	554, 555	MA(RC)
3-17	10-bar seeded sample	Mahler61	567, 568	MA(RC)
3-18	10-bar seeded sample	Mahler61	581, 582	MA(RC)
3-19	10-bar seeded sample	Mahler63	79, 80	MA(RC)
3-20	10-bar seeded sample	Mahler63	85, 86	MA(RC)
3-21	10-bar seeded sample	Mahler63	90, 91	MA(RC)
3-22	10-bar seeded sample	Mahler63	90, 91	MA(RC)
3-23	10-bar seeded sample	Mahlsy54	51, 52	MA(RC)
3-24	10-bar seeded sample	Mahlsy54	147, 148	MA(RC)
3-25	10-bar seeded sample	concatenation of MAHL64O1, MAHL64O2 and MAHL64O3	17, 18	MA(RC)
3-26	10-bar seeded sample	concatenation of MAHL64O1, MAHL64O2 and MAHL64O3	17	MA(RC)
3-27	10-bar seeded sample	concatenation of MAHL64O1, MAHL64O2 and MAHL64O3	START, 134	MA(RC)
3-28	10-bar seeded sample	concatenation of MAHL64O1, MAHL64O2 and MAHL64O3	158, 159	MA(RC)
3-29	10-bar seeded sample	concatenation of MAHL64O1, MAHL64O2 and MAH64O3	210, 211	MA(RC)
3-30	10-bar seeded sample	concatenation of MAHL64O1, MAHL64O2 and MAHL64O3	446, 447	MA(RC)
3-31	10-bar seeded sample	concatenation of MAHL64O1, MAHL64O2 and MAHL64O3	454, 455	MA(RC)
3-32	10-bar seeded sample	concatenation of MAHL64O1, MAHL64O2 and MAHL64O3	466, 467	MA(RC)
3-33	10-bar seeded sample	concatenation of MAHL64O1, MAHL64O2 and MAHL64O3	490, 491	MA(RC)

3-34	10-bar seeded sample	concatenation of MAHL64O1, MAHL64O2 and MAHL64O3	491, 492	MA(RC)
3-35	10-bar seeded sample	concatenation of MAHL64O1, MAHL64O2 and MAHL64O3	689, 690	MA(RC)
3-36	10-bar seeded sample	concatenation of MAHL64O1, MAHL64O2 and MAHL64O3	723, 724	MA(RC)
3-37	10-bar seeded sample	concatenation of MAHL64O1, MAHL64O2 and MAHL64O3	727, 728	MA(RC)
3-38	10-bar seeded sample	concatenation of MAHL64O1, MAHL64O2 and MAHL64O3	727, 728	MA(RC)
3-39	10-bar seeded sample	3rd-movement-Feierlich-und-gemessen-ohne-zu-schleppen.midi	2, 3	MA(RC)
3-40	10-bar random sample	-	-	MA(RC)
3-41	10-bar random sample	-	-	MA(RC)
3-42	10-bar random sample	-	-	MA(RC)
3-43	100-bar random sample	-	-	MA(RC)
3-44	100-bar random sample	-	-	MA(RC)
3-45	10-bar seeded sample	Mahler61	554, 555	MA(SC)
3-46	10-bar seeded sample	Mahler61	58, 59	MA(SC)
3-47	10-bar seeded sample	Mahler61	58, 59	MA(SC)
3-48	10-bar seeded sample	Mahler61	59, 60	MA(SC)
3-49	10-bar seeded sample	Mahler61	62, 63	MA(SC)
3-50	10-bar seeded sample	Mahler61	134, 135	MA(SC)
3-51	10-bar seeded sample	Mahler61	268, 269	MA(SC)
3-52	10-bar seeded sample	Mahler61	469, 470	MA(SC)
3-53	10-bar seeded sample	Mahler61	581, 582	MA(SC)
3-54	10-bar seeded sample	Mahler61	583, 584	MA(SC)
3-55	10-bar seeded sample	Mahler61	600, 601	MA(SC)
3-56	10-bar seeded sample	Mahler61	602, 603	MA(SC)
3-57	10-bar seeded sample	Mahler63	79, 80	MA(SC)
3-58	10-bar seeded sample	Mahler63	79, 80	MA(SC)
3-59	10-bar seeded sample	Mahler63	85, 86	MA(SC)
3-60	10-bar seeded sample	Mahler63	85, 86	MA(SC)
3-61	10-bar seeded sample	Mahler63	110, 111	MA(SC)
3-62	10-bar seeded sample	Mahler63	110, 111	MA(SC)
3-63	10-bar seeded sample	Mahler63	110, 111	MA(SC)
3-64	10-bar seeded sample	concatenation of MAHL64O1, MAHL64O2 and MAHL64O3	133, 134	MA(SC)

3-65	10-bar seeded sample	concatenation of MAHL64O1, MAHL64O2 and MAHL64O3	161, 162	MA(SC)
3-66	10-bar seeded sample	concatenation of MAHL64O1, MAHL64O2 and MAHL64O3	210, 211	MA(SC)
3-67	10-bar seeded sample	concatenation of MAHL64O1, MAHL64O2 and MAHL64O3	446, 447	MA(SC)
3-68	10-bar seeded sample	concatenation of MAHL64O1, MAHL64O2 and MAHL64O3	528, 529	MA(SC)
3-69	10-bar seeded sample	concatenation of MAHL64O1, MAHL64O2 and MAHL64O3	724, 725	MA(SC)
3-70	10-bar seeded sample	concatenation of MAHL64O1, MAHL64O2 and MAHL64O3	726, 727	MA(SC)
3-71	10-bar random sample	-	START	MA(SC)
3-72	10-bar random sample	-	START	MA(SC)
3-73	10-bar random sample	-	START	MA(SC)
3-74	10-bar random sample	-	START	MA(SC)
3-75	100-bar random sample	-	START	MA(SC)
3-76	100-bar random sample	-	START	MA(SC)

Extras

Name	Type	File	Bars	Dataset
4-1	10-step interpolation	Mahler61	1, 5	MA(R)
4-2	10-step interpolation	Mahler61	1, 9	MA(R)
4-3	10-step interpolation	pathetique_1	106, 111	PI(R)