

# **Composing a melody with long-short term memory (LSTM) Recurrent Neural Networks**

Konstantin Lackner



**TUM**



**Bachelor's thesis**

# **Composing a melody with long-short term memory (LSTM) Recurrent Neural Networks**

Konstantin Lackner

February 15, 2016



Institute for Data Processing  
Technische Universität München



Konstantin Lackner. *Composing a melody with long-short term memory (LSTM) Recurrent Neural Networks*. Bachelor's thesis, Technische Universität München, Munich, Germany, 2016.

Supervised by Prof. Dr.-Ing. K. Diepold and Thomas Volk; submitted on February 15, 2016 to the Department of Electrical Engineering and Information Technology of the Technische Universität München.

© 2016 Konstantin Lackner

Institute for Data Processing, Technische Universität München, 80290 München, Germany,  
<http://www.ldv.ei.tum.de>.

This work is licenced under the Creative Commons Attribution 3.0 Germany License. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/3.0/de/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.

# Contents

<b>1. Introduction</b>	<b>5</b>
<b>2. State of the Art in Algorithmic Composition</b>	<b>7</b>
2.1. Non-computer-aided Algorithmic Composition . . . . .	7
2.2. Computer-aided Algorithmic Composition . . . . .	8
<b>3. Neural Networks</b>	<b>11</b>
3.1. Feedforward Neural Networks . . . . .	11
3.1.1. Learning: The Backpropagation Algorithm . . . . .	13
3.2. Recurrent Neural Networks . . . . .	17
3.2.1. The Backpropagation Through Time Algorithm . . . . .	17
3.3. LSTM Recurrent Neural Networks . . . . .	18
3.3.1. Forward Pass . . . . .	18
<b>4. Data Representation: MIDI</b>	<b>21</b>
4.1. Comparison between Audio and MIDI . . . . .	21
4.2. Piano Roll Representation . . . . .	22
<b>5. Implementation</b>	<b>25</b>
5.1. The Training Program . . . . .	26
5.1.1. MIDI file to piano roll transformation . . . . .	26
5.1.2. Network Inputs and Targets . . . . .	28
5.1.3. Network Properties and Training . . . . .	30
5.2. The Composition Program . . . . .	31
<b>6. Experiments</b>	<b>33</b>
6.1. Train and Test Data . . . . .	33
6.2. Training of eleven Topologies and Network Compositions . . . . .	35
<b>7. Evaluation</b>	<b>37</b>
7.1. Subjective Listening Test . . . . .	37
7.1.1. Test Design . . . . .	37
7.1.2. Test Results . . . . .	39
<b>8. Conclusion</b>	<b>43</b>

*Contents*

<b>Appendices</b>	<b>45</b>
<b>A. Test Data Score</b>	<b>47</b>
<b>B. Network Compositions Score</b>	<b>49</b>
<b>C. Human Melodies Score</b>	<b>51</b>
<b>Bibliography</b>	<b>51</b>

# 1. Introduction

In the recent years the research on artificial intelligence (AI) has been increasingly progressing, mainly because of the huge amounts of generated data in basically every part of one's digital life, out of which AI algorithms can be trained very intensively and accurately. Beyond that, the progress in computation capabilities of modern hardware helped this field to flourish. So far, certain methods to implement artificial intelligence have been developed that could outperform human abilities, such as the Chess Computer DeepBlue or IBM's Watson who beat the best humans in the game "Jeopardy". One method of implementing Artificial Intelligence are Artificial Neural Networks, which have been developed motivated by how a human or animal brain works. Artificial Neural Networks have increasingly succeeded in tasks such as pattern recognition, e.g. in speech and image processing. However, when it comes to creative tasks such as music composition, only little research has been done in this area.

The subject of this thesis is to investigate the capabilities of an Artificial Neural Network to compose music. In particular, this thesis will focus on the composition of a melody to a given chord sequence. The main goal is to implement a long-short term memory (LSTM) Recurrent Neural Network (RNN), that composes melodies that sound pleasantly to the listener and cannot be distinguished from human melodies. Furthermore, the evaluation of the composed melodies plays an important role, in order to objectively assess the quality of the LSTM RNN composer and therefore be able to make a contribution to the research in this area.

This thesis is structured as follows. In chapter 2 the state of the art in the area of algorithmic composition will be discussed and a historic overview as well as prior approaches to computer-aided algorithmic compositions will be highlighted. Chapter 3 will provide an understanding about Neural Networks and LSTM Recurrent Neural Networks in particular. In chapter 4 the representation of music in the MIDI format will be explained, while chapter 5 details the implementation of the algorithm for composing a melody. The experiments that have been done with the implementation and the compositions created by the LSTM RNN will be discussed in chapter 6. Chapter 7 is about the evaluation of the computer-generated melodies by comparing them to human-created melodies in a listening test with human subjects. Finally, the main conclusions drawn from this thesis will be discussed in chapter 8.





## 2. State of the Art in Algorithmic Composition

Algorithmic Music Composition has been around for several centuries, dating back to Guido d'Arezzo in 1024 who invented the first algorithm for composing music, Nierhaus (2009). While there have been several approaches to algorithmic composition in the pre-computer era, the most prominent examples of algorithmic composition have been created by computers. Because of the tremendous capabilities a computer has to offer, algorithmic music composition could flourish from the beginning of the 1950s to the present.

### 2.1. Non-computer-aided Algorithmic Composition

This section is going to give a non-comprehensive overview about the history of algorithmic composition, showing major events that contributed to the state of the art.

- 3000BC - 1200AD - Development of symbol, writing and numeral system:  
“In order to be able to apply algorithms, the symbol must be introduced as a sign whose meaning may be determined freely, language must be put into writing and a number system must be designed”, Nierhaus (2009). Around 3000BC the first fully developed writing system can be found in Mesopotamia and Egypt, which is an essential abstraction process for algorithmic thinking. First sources for a closed number system date back to 3000BC as well, to a sexagesimal system with sixty as a base, found on clay tables of the Sumerian Empire. This system has been adopted by the Accadians and finally by Babylonians. The nowadays used Indo-Arabic number system became established in Europe only from the 13th century, Nierhaus (2009).
- Around 550BC - Pythagoras mathematically described musical harmony:  
Pythagoras is supposed to have found the correlation between consonant sounds and simple number ratios, and ultimately that music and mathematics share the same fundamental basis, Wilson (2003). Based on experiments with the “Monochord” he developed the Pythagorean scale, by taking any note and produce related ones by simple whole-number ratios. For example, a vibrating string produces a sound with frequency  $f$ , while a string of half the length vibrates with a frequency of  $2f$  and produces an octave. A string of  $\frac{2}{3}$  of the length produces a fifth with the frequency  $\frac{3}{2}f$ . Consequently, an octave is produced by a ratio of  $\frac{2}{1}$  and a fifth by a

## 2. State of the Art in Algorithmic Composition

ratio of  $\frac{3}{2}$  in regard of the base frequency  $f$ . The development of the Pythagorean tuning built a foundation for the nowadays used “Well temperament”.

- 1024AD - Guido d’Arezzo created the first technique for algorithmic composition: Besides building the foundation for our conventional notation system of music and inventing the Hexachordsystem, Guido d’Arezzo developed solmization around AD 1000 (Simoni, 2003). Solmization is a system where letters and vowels of a religious text are mapped onto different pitches, thus creating an automated way of composing a melody, Nierhaus (2009). He developed this system to reduce the time a monk needed to learn all Gregorian Chorals.
- 1650AD - Athanasius Kircher presented his Arca Musarithmetica: In his book *Musurgia Universalis* Athanasius Kircher presented the *Arca Musarithmetica*, a mechanical machine for composing music, Stange-Elbe (2015). The device consisted of a box with wooden faders to adjust different musical parameters, such as pitch, rhythm or beat. By freely combining the different faders a lot of different musical sequences could be created. By creating the Arca Musarithmetica, Kircher presented a way for composing music based on algorithmic principles, apart from any subjective influence, Stange-Elbe (2015).
- 18th century - Musical dice game: The musical dice game, which became very popular around Europe in the 18th century, is a system for composing a minuet or valse in an algorithmic manner, without having knowledge about composition. The dice game consists of two dies, a sheet of music and a look-up table. The result of the dice roll and the number of throws determine the row and column for the look-up table, which points to a certain bar within the sheet of music. The piece is composed by adding one bar from the sheet music to the composition for each dice throw, Windisch. Probably the oldest version of the dice game has been developed by the composer Johann Philipp Kirnberger, although the most popular version has been developed by W. A. Mozart.

There is a major difference in the capabilities of non-computer-aided and computer-aided Algorithmic Composition techniques. The list above gave an overview about non-computer-aided algorithmic composition approaches, while the next chapter is going to focus on computer-aided Algorithmic Music Composition.

### 2.2. Computer-aided Algorithmic Composition

For composing music with an algorithm, there are several AI (Artificial Intelligence) methods to implement such an algorithm: Mathematical Models, Knowledge based systems, Grammars, Evolutionary methods, Systems which learn and Hybrid systems, Papadopoulos. However, there are also non-AI methods such as systems based on random numbers.

## 2.2. Computer-aided Algorithmic Composition

The following gives an overview about the most prominent examples of computer-aided algorithmic composition.

- 1955 - "Illiac Suite" by Lejaren Hiller and Leonard Isaacson:  
The first completely computer-generated composition was made by Hiller and Isaacson in 1955 on the ILLIAC computer at the University of Illinois, Nierhaus (2009). The composition is on a symbolic level, that is the output of the system represents note values that must be interpreted by a musician. The "Illiac Suite" is a composition for a string quartett, which is divided into four movements, or so-called experiments. The experiments 1 and 2 make use of counterpoint techniques modeled on the concepts of Josquin de Près and Giovanni Pierluigi da Palestrina for generating musical content. Experiment 3 is composed in a similar manner, but with a less restrictive rule system. In experiment 4 markov models of variable order are used for the generation of musical structure, Hiller (1959). The Illiac Suite for string quartett was first performed in August 1956.
- 1955 - "Metastasis" by Xenakis has its world premiere:  
Iannis Xenakis had a major impact on the development of algorithmic composition. Having started his professional career as an architectural assistant, Xenakis began applying his architectural design ideas on music as well. His piece "Metastasis" for orchestra was his first musical application of this kind, using long, interlaced string glissandi to "obtain sonic spaces of continuous evolution", Dean (2009). This and further pieces of Xenakis involve the application of stochastics, markov chains, game theory, boolean logic, sieve theory and cellular automata, Dean (2009). Xenakis' works have been influenced by other pioneers in the field of algorithmic composition, such as Gottfried-Michael Koenig, David Cope or Hiller and Isaacson.
- 1981 - Experiments in Musical Intelligence (EMI) by David Cope:  
The Experiments in Musical Intelligence is a system of algorithmic composition, which generates compositions conforming to a given musical style. In EMI several different approaches for music generation are combined and it is often mentioned in the context of Artificial Intelligence, while Cope himself describes his system in the framework of a "musical turing test", Nierhaus (2009). For EMI, Cope developed the approach of musical "recombinancy", which in analogy to the musical dice game composes music by arranging musical components. However, the musical components are autonomously detected by EMI by means of the complex analysis of a corpus and they are partly transformed and recombined by EMI. The complex strategies of recombination are implemented within an augmented transition network, which is responsible for pattern matching and the reconstruction process, da Silva (2003). For Cope, EMI emulates the creative process taking place in human composers: "This program thus parallels what I believe takes place at some level in composers minds, whether consciously or subconsciously. The genius of

## 2. State of the Art in Algorithmic Composition

great composers, I believe, lies not in inventing previously unimagined music but in their ability to effectively reorder and refine what already exists”, Nierhaus (2009).

- 1994 - Mozer presents his model “CONCERT”:  
Micheal Mozer developed the system “CONCERT” that, among other things, composes melodies to underlying harmonic progressions, which is based on Recurrent Neural Networks, Nierhaus (2009). A simple algorithmic music composition approach is to select notes sequentially according to a transition table, that specifies the probability of the next note based on the previous context. Mozer adapted this system by using a recurrent autopredictive connectionist network, that has been trained on soprano voices of Bach chorales, folk music melodies and harmonic progressions of various waltzes, Mozer (1994). An integral part of CONCERT is the incorporation of psychologically-grounded representations of pitch, duration and harmonic structure. Mozer describes CONCERT’s compositions as “occasionally pleasant” and although they are preferred over compositions by third-order transition tables, they lack “global coherence”. That means that interdependencies in longer musical sequences could not be extracted and the compositions of CONCERT tend to be arbitrary.
- 2002 - Eck and Schmidhuber research in music composition with LSTM RNNs:  
Based on the CONCERT model with Recurrent Neural Networks (RNNs), Douglas Eck and Jürgen Schmidhuber developed an algorithm for composing melodies using long-short term memory (LSTM) RNNs. Since LSTM RNNs are capable of capturing interdependencies between temporary distant events, their approach should overcome CONCERT’s problem of a lack of global structure, Eck (2002). The research done by Eck and Schmidhuber consists of two experiments, where in the first one the LSTM RNN learned to reproduce a musical chord structure. This task was easily handled by the network as it could generate any number of continuing cycles, once one full cycle of the chord sequence was generated. The second experiment comprised the learning of chords and melody in the style of a blues scheme. The network compositions were remarkably better sounding than a random walk on the pentatonic scale, although they “diverge from the training set at times significantly”, Eck (2002). In an evaluation done with a jazz musician, he “is struck by how much the compositions sound like real bebop jazz improvisation over this same chord structure”, Eck (2002).

Motivated by the promising results created by Eck and Schmidhuber, the algorithm for this thesis is based on LSTM RNNs as well. The next chapter will give an introduction to Neural Networks and will highlight the advantages of LSTM Recurrent Neural Networks over vanilla Neural Networks.

### 3. Neural Networks

The following will give an introduction to Neural Networks in regard of algorithmic music composition. First, Feedforward Neural Networks and the Backpropagation algorithm will be explained. From there, Recurrent Neural Networks and LSTM Networks will be further detailed.

#### 3.1. Feedforward Neural Networks

Artificial Neural Networks have been developed motivated by how the human or animal brain works. A Neural Network “is a massively parallel distributed processor made up of simple processing units, which has a natural propensity for storing experiential knowledge and making it available for use”, (Haykin, 2004).

**Neurons** The “simple processing units” are called Neurons, which take a number of inputs, sum all inputs together and compute the Neuron’s output by squashing the sum with an activation function.

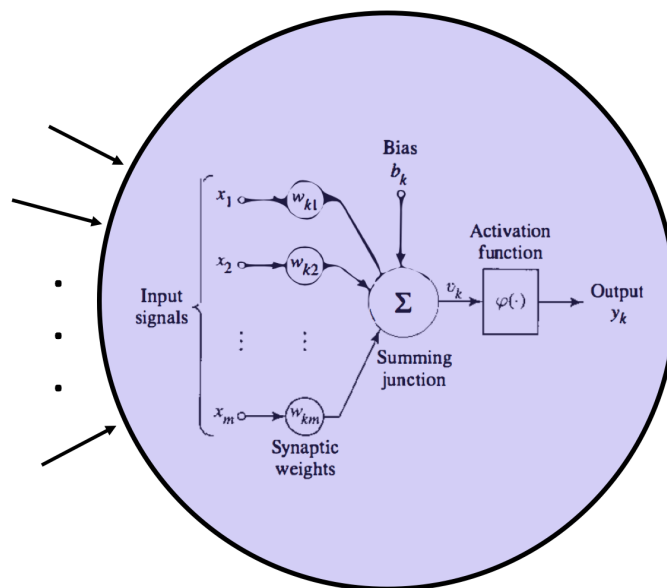


Figure 3.1.: A Neuron. Source: Haykin (2004)

### 3. Neural Networks

The input signals  $x_i$ ,  $i = 1, 2, \dots, m$ , are multiplied by a weight  $w_{ki}$  where  $k$  is referring to the number of the current Neuron and  $i$  to the number of the input signal. The net input  $v_k$  is calculated by the sum over all input signals. Besides the inputs there is also a bias  $b_k$  feeding into the net input that gives the Neuron a tendency towards a specific behaviour. The net input  $v_k$  can be calculated as follows:

$$u_k = \sum_{i=1}^m w_{ki} x_i \quad (3.1)$$

$$v_k = u_k + b_k \quad (3.2)$$

The net input  $v_k$  can also be calculated with:

$$v_k = \sum_{i=0}^m w_{ki} x_i \quad (3.3)$$

where  $x_0 = 1$  and  $w_{k0} = b_k$ .

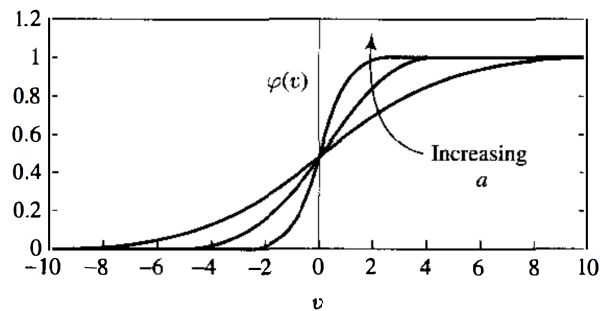
To calculate the output, also called activation,  $y_k$  of a Neuron, an activation function  $\varphi(\cdot)$  is applied on the net input  $v_k$ :

$$y_k = \varphi(v_k) \quad (3.4)$$

There are several types of activation functions used, while the *sigmoid* function is the most common one, which can be seen in equation 3.5. It squashes the net input to an output between 0 and 1.

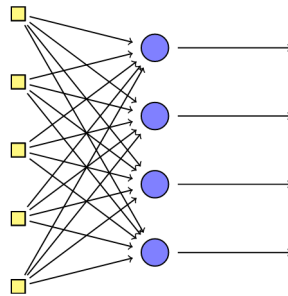
$$\sigma(v_k) = \frac{1}{1 + e^{(-a \cdot v_k)}} \quad (3.5)$$

Equation 3.5 shows the sigmoid function with  $a$  as the slope parameter. Figure 3.2 shows the graph of the sigmoid function with different values for  $a$ .



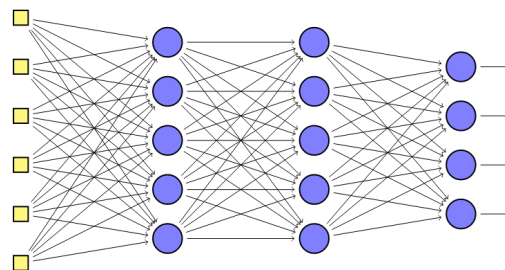
**Figure 3.2.:** Sigmoid function with different values for the slope parameter. Source: Haykin (2004)

**Network Architecture** Making a Neural Network a “massively parallel distributed processor” is achieved by arranging and connecting several Neurons to a network with a distinct architecture. The simplest architecture is called a *Single-layer Feedforward Network*, Haykin (2004). It consists of two layers of Neurons, an input and an output layer. The Neurons between both layers are fully connected through “Synapses” with the synaptic weight, Haykin (2004). Figure 3.3 shows a Single-layer Feedforward Network with five input units and four Neurons as the output units. Every input unit feeds into each Neuron of the output layer.



**Figure 3.3.:** Single-layer Feedforward network with five input units and four Neurons as the output. Source: Johnson (2015)

Another commonly used network architecture is the *Multi-layer Feedforward Neural Network*, which is similar to the Single-layer Feedforward Network but with more layers between the input and output layer, the so-called hidden layers. Through the hidden layers, a network is able to “extract higher-order statistics” and a “global perspective”, Haykin (2004). An example for a Multi-layer Feedforward Neural Network is given in figure 3.4.



**Figure 3.4.:** Multi-layer Feedforward Network with two hidden layers. Source: Johnson (2015)

### 3.1.1. Learning: The Backpropagation Algorithm

The network’s knowledge is acquired by the network through a learning process, where the synaptic weights are adjusted in a way that the network’s output matches the de-

### 3. Neural Networks

sired output, which is called *supervised learning*. The network is trained with training data  $\{(x^{(1)}, t^{(1)}), (x^{(2)}, t^{(2)}), \dots, (x^{(n)}, t^{(n)})\}$ , consisting of input values  $x^{(n)}$  and corresponding expected target values  $t^{(n)}$ , where  $n$  is referring to the number of training samples.

**Loss function** By taking the difference of expected values (target values)  $t_j$  and the network's actual output  $y_j$  when feeding it with the input training data, one gets a measure for the networks performance. A commonly used loss function is the *Squared Error* function in equation 3.6, where  $j$  refers to the  $j$ -th Neuron of the output layer and  $J$  refers to the total number of Neurons in the output layer, Ng (2012b).

$$E(w_{ki}) = \frac{1}{2} \sum_{j \in J} (y_j - t_j)^2 \quad (3.6)$$

**Gradient Descent** Learning takes place by adjusting the network's synaptic weights while finding a minimum of the loss function. This is mostly done using the *Gradient Descent* method, Rumelhart (1986). Equation 3.7 shows the update rule of gradient descent, where synaptic weights  $w_{ki}$  are usually initialized randomly at the beginning and are adjusted accordingly to equation 3.7, where  $\alpha$  is the so-called learning rate, Ng (2012b).

$$w_{ki} := w_{ki} - \alpha \frac{\partial}{\partial w_{ki}} E(w_{ki}) \quad (3.7)$$

If the initialized values of  $w_{ki}$  are close enough to the optimum, and the learning rate  $\alpha$  is small enough, the gradient descent algorithm achieves linear convergence, Bottou (2010).

**Computing Partial Derivatives** To apply gradient descent as described in equation 3.7 the partial derivatives of  $E(w_{ki})$  in regard of the weights  $w_{ki}$  must be computed. For this, two different cases will be treated where 1) the weights are connected between the last hidden layer and the output layer and 2) the weights are connected between two hidden layers.

**Weights at output layer** For case 1) the following shows the computation of the partial derivative of  $E(w_{ji})$  in regard of the weights  $w_{ji}$ , Ng (2012a):

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \frac{1}{2} \sum_{j \in J} (y_j - t_j)^2 \quad (3.8)$$

$$\frac{\partial E}{\partial w_{ji}} = (y_j - t_j) \frac{\partial}{\partial w_{ji}} y_j \quad (3.9)$$

Since the partial derivative of  $E$  is taken in regard of one specific  $w_{ji}$ , all terms of the sum except the one for the specific  $j$  will be zero. Applying chain rule onto the argument of the sum in equation 3.8 delivers equation 3.9. Since  $t_j$  is constant the value of  $\frac{\partial t_j}{\partial w_{ji}}$  is zero.



### 3.1. Feedforward Neural Networks

The output  $y_j$  of the  $j$ -th Neuron in the output layer is equal to the net input of that Neuron squashed by the activation function:

$$\frac{\partial E}{\partial w_{ji}} = (y_j - t_j) \frac{\partial}{\partial w_{ji}} \varphi(v_j) \quad (3.10)$$

Applying chain rule onto  $\frac{\partial}{\partial w_{ji}} \varphi(v_j)$  delivers:

$$\frac{\partial E}{\partial w_{ji}} = (y_j - t_j) \varphi'(v_j) \frac{\partial}{\partial w_{ji}} v_j \quad (3.11)$$

The partial derivative of the net input  $v_j$  in regard of the weight  $w_{ji}$  is simply the  $i$ -th input  $x_i$  of the Neuron.

$$\frac{\partial E}{\partial w_{ji}} = (y_j - t_j) \varphi'(v_j) x_i \quad (3.12)$$

For reasons of simplicity we will define:

$$\delta_j := (y_j - t_j) \varphi'(v_j) \quad (3.13)$$

and get as a result for the partial derivative of  $E$  in regard of the weights  $w_{ji}$  from the last hidden layer to the output layer:

$$\frac{\partial E}{\partial w_{ji}} = \delta_j x_i \quad (3.14)$$

**Weights between hidden layers** From here on case 2) will be looked at, where the weight  $w_{ki}^{(l)}$  is connected from the  $i$ -th Neuron in hidden layer  $l - 1$  to the  $k$ -th Neuron in hidden layer  $l$ . In this case we cannot omit the sum, as we did from equation 3.8 to equation 3.9 since the output  $y_j$  of every Neuron in the output layer is dependent on all weights previous to the weights at the output layer.

$$\frac{\partial E}{\partial w_{ki}} = \frac{\partial}{\partial w_{ki}} \frac{1}{2} \sum_{j \in J} (y_j - t_j)^2 \quad (3.15)$$

$$\frac{\partial E}{\partial w_{ki}} = \sum_{j \in J} (y_j - t_j) \frac{\partial}{\partial w_{ki}} y_j \quad (3.16)$$

Again applying  $y_j = \varphi(v_j)$  and the chain rule delivers:

$$\frac{\partial E}{\partial w_{ki}} = \sum_{j \in J} (y_j - t_j) \frac{\partial}{\partial w_{ki}} \varphi(v_j) \quad (3.17)$$

$$\frac{\partial E}{\partial w_{ki}} = \sum_{j \in J} (y_j - t_j) \varphi'(v_j) \frac{\partial}{\partial w_{ki}} v_j \quad (3.18)$$

### 3. Neural Networks

$$\frac{\partial E}{\partial w_{ki}} = \sum_{j \in J} (y_j - t_j) \varphi'(v_j) \frac{\partial v_j}{\partial y_k} \frac{\partial y_k}{\partial w_{ki}} \quad (3.19)$$

With  $\frac{\partial v_j}{\partial y_k} = w_{jk}$  and  $\frac{\partial y_k}{\partial w_{ki}}$  being independent of the sum we get:

$$\frac{\partial E}{\partial w_{ki}} = \frac{\partial y_k}{\partial w_{ki}} \sum_{j \in J} (y_j - t_j) \varphi'(v_j) w_{jk} \quad (3.20)$$

Applying  $y_k = \varphi(v_k)$  and similar steps on  $\frac{\partial y_k}{\partial w_{ki}}$  as before we get:

$$\frac{\partial E}{\partial w_{ki}} = \varphi'(v_k) \frac{\partial v_k}{\partial w_{ki}} \sum_{j \in J} (y_j - t_j) \varphi'(v_j) w_{jk} \quad (3.21)$$

$$\frac{\partial E}{\partial w_{ki}} = \varphi'(v_k) x_i \sum_{j \in J} (y_j - t_j) \varphi'(v_j) w_{jk} \quad (3.22)$$

Using  $\delta_j$  from equation 3.13 we get:

$$\frac{\partial E}{\partial w_{ki}} = x_i \varphi'(v_k) \sum_{j \in J} \delta_j w_{jk} \quad (3.23)$$

Again, for reasons of simplicity we will define:

$$\delta_k := \varphi'(v_k) \sum_{j \in J} \delta_j w_{jk} \quad (3.24)$$

And with that we get an expression for the partial derivative of  $E$  in regard of the weights  $w_{ki}$  between hidden layers:

$$\frac{\partial E}{\partial w_{ki}} = x_i \delta_k \quad (3.25)$$

**The Backpropagation algorithm** With the equations 3.14 and 3.25 above we can now formulate the Backpropagation learning algorithm on a fixed training data set  $\{(x^{(1)}, t^{(1)}), (x^{(2)}, t^{(2)}), \dots, (x^{(n)}, t^{(n)})\}$ , where  $x^{(n)}$  is a vector of input values,  $t^{(n)}$  is a vector of target values and  $n$  is the number of training samples, Ng (2012a).

1. For  $b = 1$  to  $n$ :

- a) Feed forward through the net with input values  $x^{(b)}$
- b) For the output layer, compute  $\delta_j$
- c) Backpropagate the error by computing  $\delta_k$  for all layers previous to the output layer

- d) Compute the partial derivatives for the output layer  $\frac{\partial E}{\partial w_{ji}} = \delta_j x_i$  and for all hidden layers  $\frac{\partial E}{\partial w_{ki}} = x_i \delta_k$ .
- e) Use gradient descent to update the weights  $w_{ki} := w_{ki} - \alpha \frac{\partial}{\partial w_{ki}} E(w_{ki})$

As we have seen, the network is learning to output the desired values by adjusting its weights. Therefore the knowledge of a Neural Network is stored in the network's weights, Haykin (2004). There are several methods available for training a Neural Network such as adaptive step algorithms or second-order algorithms, Rojas (1996), while the above described Backpropagation algorithm is one of the most popular ones.

With the above described architecture of a Multi-layer Neural Network and an appropriate learning algorithm several tasks can be achieved, for example handwriting recognition, object recognition in image processing or spectroscopy in the field of chemistry, Svozil (1997). Although Multi-layer Neural Networks are achieving good results on those tasks, they lack the ability to capture patterns over time, which is key for music composition. Recurrent Neural Networks are a special type of Neural Networks that can capture information over time.

## 3.2. Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are able to capture time dependencies between inputs. In order to do that, the output of Neurons is fed back into its own input and inputs of other Neurons in the next time step. By that, information of previous time steps is being captured and influencing the computation process.

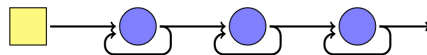


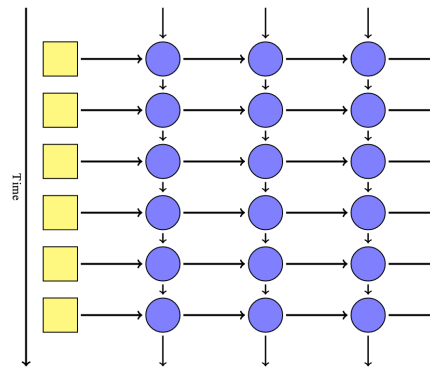
Figure 3.5.: Simple RNN structure. Source: Johnson (2015)

By unfolding the time axis, Figure 3.5 can also be represented as follows:

### 3.2.1. The Backpropagation Through Time Algorithm

Since the network architecture has changed, the learning algorithm also needs to be adapted. For recurrent networks an adapted version of the "Backpropagation" Algorithm from section 3.1.1 is mostly being used, the so-called "Backpropagation through time" Algorithm, Lipton (2015). By unfolding a RNN in time a Feedforward Network is produced,

### 3. Neural Networks



**Figure 3.6.:** Simple RNN structure. Source: Johnson (2015)

provided the network is fed with finite time steps, Principe (1997). This can also be seen in figure 3.6. When having an unfolded RNN, the backpropagation algorithm from section 3.1.1 can be applied to train the RNN.

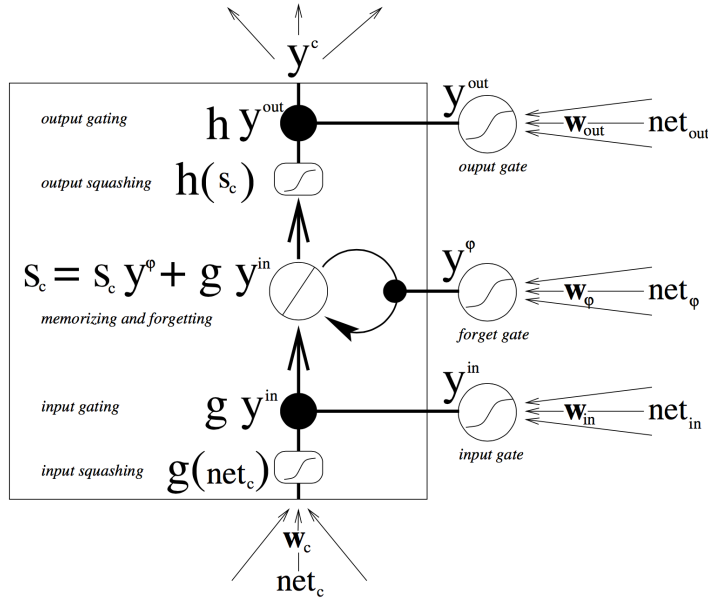
Research by Mozer found out that for music composed with RNNs the “local contours made sense” but “the pieces were not musically coherent”, Eck (2002). Therefore Eck suggested to use long short-term memory Recurrent Neural Networks (LSTM RNNs) which will be explored in the next chapter, Eck (2002).

### 3.3. LSTM Recurrent Neural Networks

LSTM RNNs (long short-term memory Recurrent Neural Networks) are a special kind of Recurrent Neural Networks designed to avoid the “rapid decay of backpropagated error”, Gers (2001). In a LSTM RNN the Neurons are replaced by a “Memory Block” which can contain several “Memory Cells”. Figure 3.7 shows such a Memory Block containing one Memory Cell. The input of a memory block can be gated via the “Input Gate”, the output can be gated via the “Output Gate”. Each memory cell has a recurrent connection which can also be gated via the “Forget Gate”. The three gates can be seen as a read, write and reset functionality as in common memories.

#### 3.3.1. Forward Pass

The description of the forward pass is taken from Gers (2001), who has introduced LSTM RNNs for the first time with its current functionalities. The current state  $s_c$  of a Memory Cell is based on its previous state, on the cell’s net input  $net_c$  and on the Input Gate’s net input  $net_{in}$  as well as the Forget Gate’s net input  $net_{\phi}$ :



**Figure 3.7.:** The LSTM Memory Block replaces the Neurons of vanilla Recurrent Neural Networks. Source: Gers (2001)

$$s_c = s_c y^\phi + g(\text{net}_c) y^{\text{in}} \quad (3.26)$$

The cell's net input  $\text{net}_c$  is squashed by an activation function  $g(\cdot)$  and then multiplied by  $y^{\text{in}}$ , which is computed with:

$$y^{\text{in}} = \sigma(\text{net}_{\text{in}}) \quad (3.27)$$

where  $\sigma(\cdot)$  refers to the sigmoid function (eq. 3.5). By multiplying  $g(\text{net}_c)$  with  $y^{\text{in}}$ , the Input gate can prevent the cell's state to be updated by its net input  $\text{net}_c$ , if  $y^{\text{in}} = 0$ . The cell's state can also be forgotten with the Forget Gate, if  $y^\phi = \sigma(\text{net}_\phi) = 0$ .

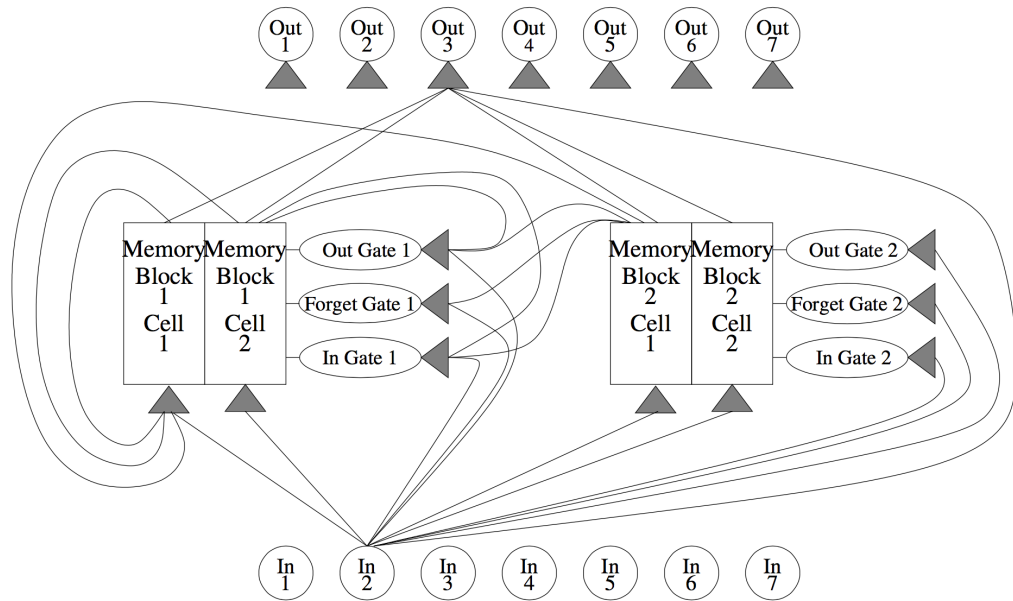
The cell's output  $y^c$  is computed by squashing the cell's state  $s_c$  with  $h(\cdot)$  and multiplying it with the Output Gate's output  $y^{\text{out}} = \sigma(\text{net}_{\text{out}})$ :

$$y^c = h(s_c) y^{\text{out}} \quad (3.28)$$

Figure 3.8 shows how Memory Blocks are integrated into a LSTM RNN Network. LSTM Networks can also be trained by the Backpropagation Through Time Algorithm from section 3.2.1, Gers (2001).

Because of the capability to capture dependencies between long-distant timesteps, which is necessary to abstract the characteristics of music, LSTM Recurrent Neural Networks have been chosen for the composition of a melody. Since LSTM RNNs need to be

### 3. Neural Networks



**Figure 3.8.:** An example of a LSTM Network. For simplicity, not all connections are shown. Source: Gers (2001)

fed and trained with numeric data, an abstraction of a musical melody is necessary. The next chapter will elaborate on the data representation of a melody, that has been chosen for this thesis.

## 4. Data Representation: MIDI

In the previous chapter we have seen what Neural Networks are and that a LSTM RNN is the most promising type to use when it comes to music composition. For training and using an LSTM RNN the question arises how music is going to be represented, in order to make it accessible for the Neural Network. One possible option is to use vanilla audio data, such as wave files, to feed the Neural Net. Another option is to use MIDI data, which does not contain any audible sound, but information about the score of a musical piece. The next section will compare these two options and come to the conclusion to use MIDI data for the implementation of the algorithm.

### 4.1. Comparison between Audio and MIDI

To decide whether Audio or MIDI data is the right choice to use, it is necessary to ask for the purpose of the Neural Network implementation. In this case, the purpose of the LSTM Network is to compose a melody or in other words find a melody to a given chord sequence. To reduce the complexity of this task, we are only interested in the pitch, the start and the length of the melody's notes. The velocity or other forms of articulation such as bending will not be considered as part of this thesis.

**Audio** An audio signal is a very rich representation of music, since it can capture almost every detail of music, depending on the audio format and quality. For example, audio signals contain the timbre of instruments, which is the characteristic spectrum of an instrument, its characteristic transients as well as the development of the spectrum over time, Levitin (2006). To reduce the complexity of an audio signal to just the pitch, the start and the length of the notes in a melody, rather complex methods have to be applied. For example, to extract the pitch of a note a Fourier Transform is necessary to detect the base frequency of this tone which then need to be mapped to a specific pitch, which also is a nonlinear function, Zwicker (1999). To extract the start of a note the transients would have to be detected with a Beat Detection algorithm and then need to be mapped to a timestep of the network. This shows that it is a rather complex undertaking to extract the necessary features for the Neural Network model used in this thesis.

**MIDI** MIDI (Musical Instrument Digital Interface) is a standardized data protocol to exchange musical control data between digital instruments. Nowadays it is mostly being used

#### 4. Data Representation: MIDI

in the context of computer music, where the actual sound is created by instruments or synthesizers in the computer. MIDI data is fed into a synthesizer with the information about a note's start, duration, and pitch. In addition there are several other options to control a digital instrument with MIDI data, which are not relevant for this thesis. MIDI data already contains the necessary information needed to feed the Neural Network and it only needs to be transformed into an appropriate numeric representation for the LSTM RNN. Thus, MIDI data has been chosen to represent music on a very basic level: pitch, start and length of notes. The following chapter elaborates how MIDI data will be transformed to make it accessible for the Neural Network.

### 4.2. Piano Roll Representation

The necessary information as part of this thesis is a note's pitch, start time and length only. To represent the incoming MIDI data in a manner that only this information is feeding the LSTM RNN, a piano roll representation has been chosen. A piano roll shows on the vertical axis chromatically the notes as on a piano keyboard and the horizontal axis displays time. For the time a note is played, a bar with the length and the pitch of the note is denoted in the piano roll. Figure 4.1 shows an example of a piano roll representation of a chord sequence, which score can be seen in figure 4.2.



Figure 4.1.: Piano Roll Representation of the Score in Figure 4.2.



Figure 4.2.: Score of a twelve bar long chord sequence. Source: Eck (2002)



## 4.2. Piano Roll Representation

The piano roll representation is transformed to a two-dimensional matrix with pitch as the first dimension and time as the second dimension. Time is quantized in MIDI ticks, where the default setting is 96 ticks per beat and one beat typically refers to a quarter note (2015). 96 ticks per beat lead to a resolution of a  $\frac{1}{384}$ -th note per timestep, which is far too granular for the purposes of this thesis, since the melodies used for this thesis contain no shorter notes than  $\frac{1}{16}$ -th notes. To reduce the computation costs, the number of ticks per beat needs to be reduced. 4 ticks per beat lead to a resolution of a  $\frac{1}{16}$ -th note per timestep and the number of  $\frac{1}{16}$ -th quantization steps in a MIDI file determines therefore the size of the time axis. The size of the pitch axis is dependent on the note range of a piece. All pitches below the lowest note and all pitches higher than the highest note will be neglected. Therefore, the piano roll matrix is of the size (*num of  $\frac{1}{16}$ -th steps, note range*). If a note from the piano roll is being played at one particular tick, this will be denoted with a 1 in the matrix at this tick and the note's pitch. If a note is not being played, this will be denoted with a 0 in the matrix. Figure 4.3 shows the matrix for the first four bars of the piano roll in figure 4.1.

```
[ 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0.]
[ 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0.]
[ 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0.]
[ 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0.]
[ 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0.]
[ 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0.]
[ 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0.]
[ 1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0.]
[ 1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0.]
[ 1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0.]
[ 1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0.]
[ 1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0.]
[ 1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0.]
[ 1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0.]
[ 1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0.]
[ 1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0.]
[ 1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0.]
[ 1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0.]
[ 1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0.]
[ 1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0.]
[ 1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0.]
[ 1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0.]
[ 1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0.]
[ 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 1. 0.]
[ 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 1. 0.]
[ 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 1. 0.]
[ 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 1. 0.]
[ 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0.]
[ 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0.]
[ 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0.]
[ 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0.]
```

**Figure 4.3.:** First four bars of the piano roll in figure 4.1 represented in matrix notation. Resolution is 4 ticks per beat.

Figure 4.3 reveals the problem that there is no distinction between several notes being played right after each other at the same pitch and one long note of the same pitch. For example, in the first three bars the note C is being played with the length of a half note. In the matrix representation however, this is represented by a “1” 24-times after each other in the column representing the note C. This could also be interpreted as C being played for the length of a one and a half note. Therefore, the ending of a note also has to be

#### 4. Data Representation: MIDI

represented. To achieve this, only the first half of the length of a note will be denoted with 1, the other half will be denoted with 0. This representation can be seen in figure 4.4 for the first four bars of the piano roll from figure 4.1.

```
[ 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0.]
[ 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0.]
[ 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0.]
[ 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0.]
[ 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[ 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[ 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[ 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[ 1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0.]
[ 1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0.]
[ 1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0.]
[ 1. 0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 0.]
[ 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[ 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[ 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[ 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[ 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0.]
[ 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0.]
[ 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0.]
[ 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0.]
[ 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[ 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[ 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[ 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 1. 0.]
[ 0. 0. 1. 0. 0. 1. 0. 1. 0. 0. 1. 0.]
[ 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[ 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[ 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0.]
[ 1. 0. 0. 0. 1. 0. 0. 1. 0. 0. 0. 0.]
[ 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[ 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

**Figure 4.4.:** First four bars of the piano roll in figure 4.1 represented in matrix notation, where the end of a note is also represented. Resolution is 4 ticks per beat.

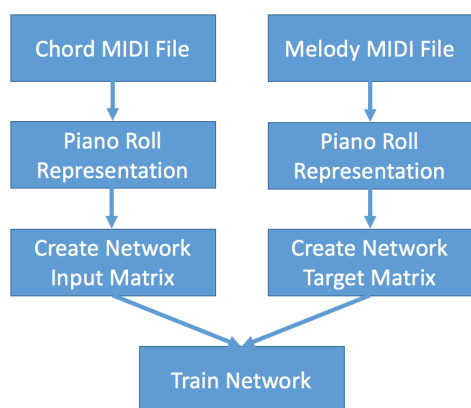
The representation of a note's end leads to a reduction of the timestep resolution, as at least two timesteps are needed to represent one note (one timestep with 1 and the other one with 0). With 4 ticks per beat, this would lead to a maximum resolution of an eighth note. In order to still achieve a maximum resolution of a sixteenth note, the number of ticks per beat is set to 8 ticks per beat for the purposes of this thesis.

It has now been described how music will be represented in the form of a piano roll matrix consisting of ones if a note is "on" and zeros if a note is "off". The next chapter will elaborate on the implementation of the data representation and the LSTM Recurrent Neural Network.

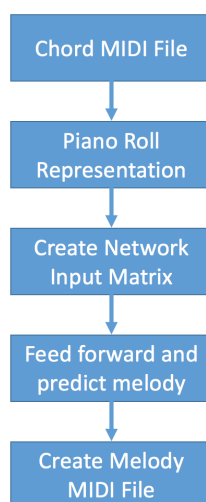
## 5. Implementation

For reasons of fast implementation the programming language “Python” has been chosen, since there exist several Neural Network and MIDI libraries for Python. For implementing the LSTM RNN the library “Keras” has been chosen, which is a library built on “Theano”. Theano is another python library that allows for fast optimization and evaluation of mathematical expressions, which is often used in Neural Network applications. While Theano allows for higher modularity and customization of a Neural Network implementation, it is also more complex, thus involves a steeper learning curve. At the same time, Keras is less modular and comes along with a few constraints, but allows the user to implement a Neural Network very easily and quickly. Therefore, due to time constraints of this thesis, Keras has been chosen as the framework for implementing the LSTM RNN.

The library “Mido” has been used to access the MIDI data and transform it into useable data for the Neural Network. Mido allows for easy access to each MIDI message, which have been used to create a piano roll representation of the MIDI file (see section 5.1.1).



**Figure 5.1.:** Basic structure for training the LSTM RNN.



**Figure 5.2.:** Basic structure for composing a melody to a new chord sequence.

The implementation has been divided into two programs. The first program is used for training the LSTM Recurrent Neural Network, the second one for composing a melody to a new chord sequence. The basic structure for training the Neural Network is shown in

## 5. Implementation

figure 5.1 and for composing a melody in figure 5.2. The following will elaborate on the implementation of the training program as well as of the composition program.

### 5.1. The Training Program

For the training of the LSTM RNN train data is necessary, that consists of chord sequences as the input and belonging melodies as the target. The goal during composition is to output a melody once the network is fed forward with a chord sequence, so the LSTM RNN needs to abstract which melodies fit to certain chord sequences, based on the training set. The chord sequences and belonging melodies need to be available as MIDI files in order to be transformed into a piano roll representation. To give a better understanding of the data transformation, in this section the data flow will be shown exemplary with the chord sequence and belonging melody given in figure 5.3.



**Figure 5.3.:** Two chords (F-clef) and belonging melody (G-clef) to exemplify the data flow in section 5.1.

#### 5.1.1. MIDI file to piano roll transformation

Since the LSTM RNN needs to be trained with numeric values, the goal is to create a piano roll representation of the MIDI files, as described in section 4.2. The MIDI messages are extracted from the MIDI files with the library “Mido” (see figure 5.4 and 5.5). The relevant information contained in the MIDI messages is:

1. A note’s pitch: The pitch is given by the “note” information. For example “note=48” refers to the pitch C4.
2. The note’s start given by the type-field “note\_on” and the value of the time-field.
3. The note’s end given by the type-field “note\_off” and the value of the time-field

The time-field is showing its values in MIDI ticks quantized at 96 MIDI ticks per beat (one beat = quarter note). It needs to be noted that the time-field of the MIDI messages is showing the values relative to each other and no absolute time values. That is, the current, absolute time position is calculated by taking the sum of MIDI ticks from the first MIDI message’s time-field up to the current MIDI message’s time-field.

**Creating the piano roll matrix** The first dimension (rows) of the piano roll matrix represents time, quantized in 8 MIDI ticks per beat, the second dimension (columns) represents the pitch. The first column refers to the note with the lowest pitch in the MIDI file, the last column to the highest pitch. With the information about a note's pitch, start time and end time, the piano roll matrix is filled with ones for the first half of the duration of the note and zeros for the second half to denote the end of a note.

Figure 5.4 and 5.5 show the incoming MIDI messages for the chord sequence and melody from figure 5.3. The piano roll representation created from the MIDI messages can be seen in figure 5.6 and 5.7.

```
note_on channel=0 note=48 velocity=88 time=0
note_on channel=0 note=52 velocity=96 time=0
note_on channel=0 note=55 velocity=101 time=0
note_off channel=0 note=48 velocity=127 time=96
note_on channel=0 note=48 velocity=102 time=0
note_on channel=0 note=51 velocity=110 time=0
note_off channel=0 note=52 velocity=127 time=0
note_on channel=0 note=53 velocity=106 time=0
note_off channel=0 note=55 velocity=127 time=0
note_on channel=0 note=57 velocity=100 time=0
note_off channel=0 note=48 velocity=127 time=96
note_off channel=0 note=51 velocity=127 time=0
note_off channel=0 note=53 velocity=127 time=0
note_off channel=0 note=57 velocity=127 time=0
```

**Figure 5.4.:** Incoming MIDI messages for the chord sequence of figure 5.3. The incoming MIDI messages are quantized at 96 MIDI ticks per beat.

```
note_on channel=0 note=60 velocity=120 time=0
note_off channel=0 note=60 velocity=127 time=48
note_on channel=0 note=62 velocity=105 time=0
note_off channel=0 note=62 velocity=127 time=48
note_on channel=0 note=65 velocity=116 time=0
note_on channel=0 note=60 velocity=115 time=24
note_off channel=0 note=65 velocity=127 time=0
note_off channel=0 note=60 velocity=127 time=72
```

**Figure 5.5.:** Incoming MIDI messages for the melody in figure 5.3. The incoming MIDI messages are quantized at 96 MIDI ticks per beat.

```
[ 1.  0.  0.  0.  1.  0.  0.  1.  0.  0.]
[ 1.  0.  0.  0.  1.  0.  0.  1.  0.  0.]
[ 1.  0.  0.  0.  1.  0.  0.  1.  0.  0.]
[ 1.  0.  0.  0.  1.  0.  0.  1.  0.  0.]
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
[ 1.  0.  0.  1.  0.  1.  0.  0.  0.  1.]
[ 1.  0.  0.  1.  0.  1.  0.  0.  0.  1.]
[ 1.  0.  0.  1.  0.  1.  0.  0.  0.  1.]
[ 1.  0.  0.  1.  0.  1.  0.  0.  0.  1.]
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
```

**Figure 5.6.:** Piano Roll Matrix representing the chord sequence in figure 5.3. The time dimension (rows) is quantized at 8 MIDI ticks per beat.

```
[ 1.  0.  0.  0.  0.  0.]
[ 1.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  0.]
[ 0.  0.  1.  0.  0.  0.]
[ 0.  0.  1.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  1.]
[ 0.  0.  0.  0.  0.  1.]
[ 0.  0.  0.  0.  0.  0.]
[ 1.  0.  0.  0.  0.  0.]
[ 1.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  0.]
```

**Figure 5.7.:** Piano Roll Matrix representing the melody in figure 5.3. The time dimension (rows) is quantized at 8 MIDI ticks per beat.

## 5. Implementation

### 5.1.2. Network Inputs and Targets

So far it has been shown how the transformation from a MIDI file to a piano roll representation has been implemented. However, in order to make the data useable for the Keras framework, the piano roll representations need to be transformed into a *Network Input Matrix* and a *Prediction Target Matrix*. Both matrices consist of training samples, where in the case of the Network Input Matrix one *network input sample* consists of a 2-dimensional input matrix and in the case of the Prediction Target Matrix one *target sample* consists of a 1-dimensional target vector.

**Creating one training sample pair** During training several timesteps from the piano roll representation of the chord sequence, the network input sample, will feed forward through the network and the network will then output a vector. Training takes place by adjusting the LSTM RNN's weights, with the goal to make the output vector's values close to the ones of the target vector (see section 3.1.1). The target vector consists of one timestep from the piano roll representation of the melody, where one timestep corresponds to one row of the piano roll matrix. The amount of timesteps from the chord sequence that will feed forward is defined by the sequence length  $n$ . So the first network input sample, that will feed forward is created by taking the first  $n$  timesteps of the chord piano roll matrix. The target vector is created by taking the  $n + 1$  timestep of the melody piano roll matrix. The first training sample pair can be seen in figure 5.8 (network input sample) and 5.9 (target sample), where the sequence length has been set to  $n = 8$ .

```
[ [ 1.  0.  0.  0.  1.  0.  0.  1.  0.  0.]  
  [ 1.  0.  0.  0.  1.  0.  0.  1.  0.  0.]  
  [ 1.  0.  0.  0.  1.  0.  0.  1.  0.  0.]  
  [ 1.  0.  0.  0.  1.  0.  0.  1.  0.  0.]  
  [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]  
  [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]  
  [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]  
  [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
```

```
[ 0.  0.  0.  0.  0.  1.]
```

**Figure 5.8.:** The first network input sample created by taking the first  $n = 8$  timesteps from the chord piano roll representation (see figure 5.6).

**Figure 5.9.:** The first target vector created by taking the  $n + 1$  timestep from the melody piano roll representation (see figure 5.7)

**Requirements for the Keras framework** The Keras framework requires to supply the LSTM RNN for training with a 3-dimensional Input Matrix of size (*number of samples*, *timesteps*, *input dimension*) and a 2-dimensional Target Matrix of size (*number of samples*, *output dimension*). “timesteps” refers here to the number of timesteps that will feed forward through the network, which is given by the value of the sequence length  $n$ . “input dimension” refers to the number of input nodes of the LSTM RNN, which corresponds to the pitch range of the chord sequence. Analogous, “output dimension” corresponds to the pitch range of the melody. For training with the Keras framework it will be supplied with the

Network Input Matrix of size (*number of samples, sequence length, chord pitch range*) and the Prediction Target Matrix of size (*number of samples, melody pitch range*). The number of samples is given by the difference between the number of timesteps of the piano roll matrix and the sequence length:  $number\ of\ samples = number\ of\ timesteps_{piano\ roll} - sequence\ length$ .

**Creating the Network Input Matrix and Prediction Target Matrix** The Network Input Matrix is created by taking one sample of size (*sequence length, chord pitch range*) from the beginning of the chord piano roll matrix. The following samples are created by shifting this “window” of size (*sequence length, chord pitch range*) timestep by timestep through the chord piano roll matrix. This is done until the window includes the timestep previous to the last timestep of the chord piano roll matrix. The Prediction Target Matrix consisting of the target vectors is created by taking the melody piano roll matrix without the first  $n$  timesteps, where  $n$  is referring to the sequence length. Figure 5.10 and 5.11 show the Network Input Matrix and the Prediction Target Matrix, that have been created out of the chord sequence and melody from figure 5.3.

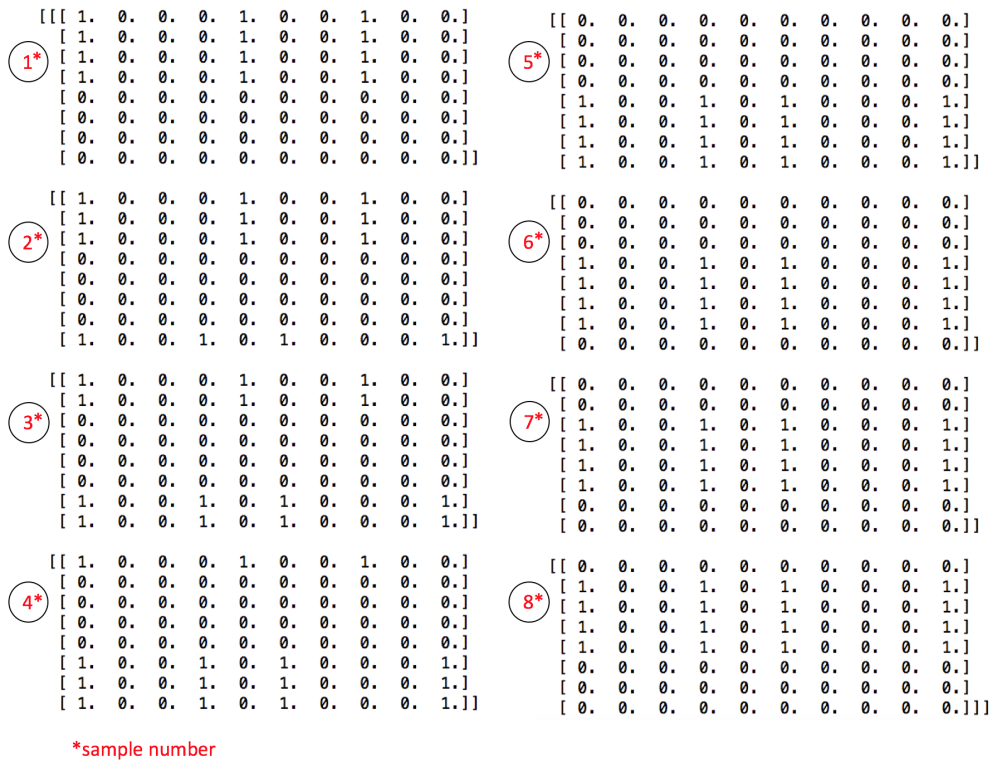


Figure 5.10.: Network Input Matrix created from the chord piano roll in figure 5.6.

## 5. Implementation

①	[	0.	0.	0.	0.	0.	1.]
②	[	0.	0.	0.	0.	0.	0.]
③	[	1.	0.	0.	0.	0.	0.]
④	[	1.	0.	0.	0.	0.	0.]
⑤	[	1.	0.	0.	0.	0.	0.]
⑥	[	0.	0.	0.	0.	0.	0.]
⑦	[	0.	0.	0.	0.	0.	0.]
⑧	[	0.	0.	0.	0.	0.	0.]

sample number

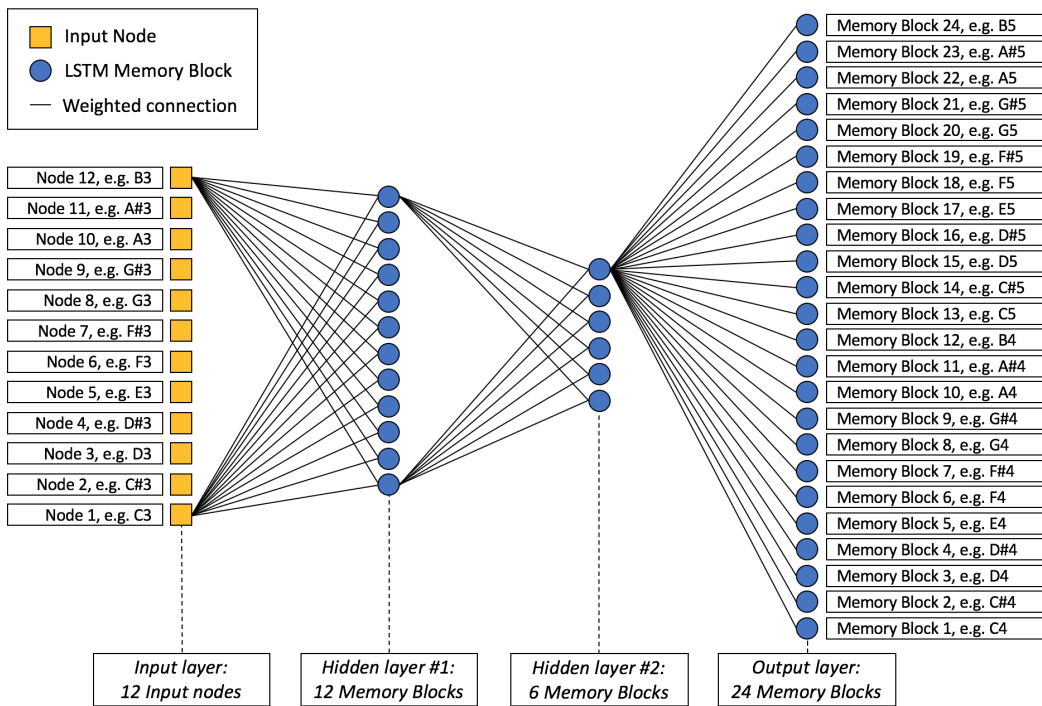
**Figure 5.11.:** Prediction Target Matrix created from the melody piano roll in figure 5.7.

### 5.1.3. Network Properties and Training

So far it has been explored how a chord and melody MIDI file are transformed into the Network Input Matrix and the Prediction Target Matrix. Once those matrices are supplied to the framework Keras, it will automatically handle the training process. Training time and the resulting performance of the LSTM RNN are heavily dependent on the Network Topology, which is detailed in the following.

**Network Topology** The LSTM RNN consists of an input layer, an output layer and optionally hidden layers between the input and output layer. The input layer consists of input nodes, that are fully connected to the subsequent layer. In the case of a 1-layer architecture (no hidden layers), the subsequent layer is the output layer, which consists of LSTM memory blocks (see section 3.3). Each input node of the input layer and each LSTM memory block of the output layer is dedicated to one specific pitch, separated in semi-tones. For reasons of keeping the computation costs at a moderate level it has been decided to limit the number of input nodes to 12 and the number of LSTM memory blocks at the output layer to 24. As a result, the chord sequences need to be within one octave and the belonging melodies within two octaves. The implemented Training Program allows to set the number of hidden layers and the number of LSTM Memory Blocks of each hidden layer to an arbitrary amount. By that, the network can be of any size chosen by the user before the training process, thus making it easy to train on different network topologies and comparing their performance. The only limitation is given by the number of input nodes in the input layer and LSTM Memory Blocks in the output layer, as stated above. Figure 5.12 shows an example of a possible network topology with two hidden layers, consisting of 12 LSTM Memory Blocks in the first hidden layer and 6 LSTM Memory Blocks in the second one.





**Figure 5.12.:** One possible Network Topology. The number and size of hidden layers can be defined by the user. For reasons of simplicity not all connections between nodes and LSTM Memory Blocks have been drawn.

## 5.2. The Composition Program

Once a LSTM Recurrent Neural Network has been trained it can be used to compose a melody. In order to do that, it needs to be fed with a chord sequence and will then output a Prediction Matrix, which can be transformed into a piano roll matrix and finally into a melody MIDI file.

At the beginning of the composition process a chord sequence within a pitch range of an octave needs to be available as a MIDI file. This MIDI file will be transformed into a piano roll representation (see section 5.1.1), which will then be transformed into a Network Input Matrix (see section 5.1.2). Keras will start predicting output values by feeding forward the samples from the Network Input Matrix, which is the actual composition process of the LSTM RNN. At the end of the composition process Keras outputs a Prediction Matrix, which consists of values between zero and one (see figure 5.13). As a next step, the Prediction Matrix is transformed into a piano roll matrix (see figure 5.14). This is done by iterating through each timestep (row) of the Prediction Matrix and finding the highest value within that timestep. If this value is higher than a certain threshold, which can be

## 5. Implementation

defined by the user, it will be replaced by a one. All other entries of one timestep will be set to zero. If the highest value of one timestep is below the threshold, all entries of the timestep will be set to zero. As a result, the piano roll matrix is representing a unisonous melody, composed by the LSTM RNN. As a consequence, the LSTM RNN needs to be trained with unisonous melodies as well. In the final step, the piano roll matrix is used to create MIDI messages in a reversed manner as the creation of the piano roll matrix from MIDI messages (see section 5.1.1). The created MIDI messages are then used to save the composed melody as a MIDI File, which concludes the whole composition process. An example for the Prediction Matrix, the piano roll matrix derived from it and the resulting score of the melody can be seen in the figures 5.13, 5.14 and 5.15. It has to be noted that for this example the pitch range has been set to six, while in the implementation the pitch range of the composed melody is 24.

```
[ 0.01  0.4   0.1   0.65  0.32  0.19]
[ 0.49  0.23  0.56  0.8   0.3   0.54]
[ 0.06  0.31  0.3   0.05  0.16  0.34]
[ 0.38  0.14  0.33  0.21  0.26  0.45]
[ 0.71  0.01  0.19  0.33  0.43  0.44]
[ 0.92  0.12  0.22  0.55  0.13  0.1 ]
[ 0.36  0.4   0.5   0.29  0.12  0.32]
[ 0.35  0.44  0.3   0.55  0.5   0.12]
[ 0.27  0.18  0.04  0.39  0.41  0.63]
[ 0.35  0.01  0.26  0.01  0.05  0.09]
[ 0.14  0.39  0.51  0.84  0.28  0.39]
[ 0.1   0.19  0.23  0.38  0.57  0.6 ]
[ 0.61  0.12  0.44  0.52  0.09  0.33]
[ 0.93  0.13  0.08  0.43  0.1   0.17]
[ 0.3   0.52  0.32  0.44  0.23  0.5 ]
[ 0.    0.01  0.13  0.58  0.32  0.1 ]
```

**Figure 5.13.:** Prediction Matrix for a melody, that will be created once the trained LSTM RNN is fed forward with a chord sequence.

```
[ 0.  0.  0.  1.  0.  0.]
[ 0.  0.  0.  1.  0.  0.]
[ 0.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  0.]
[ 1.  0.  0.  0.  0.  0.]
[ 1.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  1.]
[ 0.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  1.  0.  0.]
[ 0.  0.  0.  0.  0.  0.]
[ 1.  0.  0.  0.  0.  0.]
[ 1.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  0.]
[ 0.  0.  0.  0.  0.  0.]
```

**Figure 5.14.:** Piano Roll Matrix that has been created out of the Prediction Matrix from figure 5.13. The threshold has been set to 0.6.



**Figure 5.15.:** Score of the melody that has been composed by the LSTM RNN. The score is derived from the Piano Roll Matrix from figure 5.14.

## 6. Experiments

In section 5 the implementation of the LSTM Melody Composer has been described. The following will detail the experiments, that have been done with the Training and Composition Program. First, the train and test data will be described, that is the chords and melodies that have been used to train the LSTM RNN and the chords, that have been used for composing a melody. The train data has been used to train on eleven different network topologies. This procedure will be described in section 6.2 as well as the melodies that have been composed by the different networks. Finally, a decision on one network topology has been made that composes the most promising melodies for evaluating the quality of the compositions, which will be further explained in chapter 7.

### 6.1. Train and Test Data

The selection for the train data is a crucial part of the quality of the LSTM RNN's compositions. The LSTM RNN is deriving its "knowledge" about music from the train data only, so the better the train data, the better the LSTM RNN will perform. Also, a compositional goal needs to be defined in order to make a decision on the train data. For example, if the goal is to compose atonal melodies, the train data must consist of atonal melodies only. The test data should usually be different from the train data in order to see whether the LSTM RNN was able to make an abstraction of the train data and compose new melodies instead of simply creating a reproduction of the train data. The train and test data for the purposes of this thesis will be described in the following.

**Train Data** There are two compositional goals for the LSTM RNN. First, the LSTM RNN should compose melodies that cannot be recognized as computer compositions and therefore cannot be distinguished from melodies composed by humans. Second, the melodies should simply sound pleasantly to the listener. While the first goal requires to use melodies that have been composed by humans, it is not easy to derive a requirement for the test data from the second goal. The composed melodies should sound pleasant to the listener, but people have different tastes in music. A melody could sound very pleasant to one listener, but displeasing to another one. Therefore a decision for the train data has been made based on music that is appealing to a large group of people, namely songs from "The Beatles". Since The Beatles have been one of the most successful bands in history and their music is still influencing the creations of several musicians from today, creating

## 6. Experiments

the train data from Beatles songs seems to be a good representation for music that sounds pleasantly to a large group of people.

Another requirement is to compose melodies of a rather short duration, in order to make them usable for the listening test in the evaluation part. Although the LSTM RNN is able to compose melodies of any length, the melodies for the evaluation should be of a length of 8 bars. Therefore, the chords and melodies in the train data have been set to a length of 8 bars.

The basis for creating the train data are 16 different Beatles songs from the music book “Pop Classics For Piano: The Very Best Of The Beatles - Easy Arrangements for Piano” by Hans-Günter Heumann. The songs in this music book are already separated into a chord and melody arrangement for piano, which makes it easy to derive the train data from these songs. From the 16 songs 68 different, 8 bar long parts have been selected and transformed into MIDI files. As a result the train data consists of 68 MIDI files containing the chord sequences and 68 MIDI files containing the belonging melodies to the chords. One of the 68 chord/melody pairs can be seen in the figures 6.1 and 6.2. The pitch range of the chord sequences is between C2 and B2, while the pitch range of the melodies is allowed to be between C3 and B4. All MIDI notes have been quantized at a resolution of a sixteenth note, since this is the shortest note that can still be represented by the piano roll transformation.



**Figure 6.1.:** The chords of one of the 68 chord/melody pairs that have been used for training. The melody belonging to the chords can be seen in figure 6.2.



**Figure 6.2.:** The melody belonging to the chords of figure 6.1. It is the melody of one of the 68 chord/melody pairs that have been used for training.

**Test Data** The test data are chord sequences based on which the LSTM RNN will compose melodies. These chord sequences should be different from the chord sequences that have been used during training, in order to test the LSTM RNN's performance on unseen input data. However, it is interesting to see if there is a qualitative difference of the LSTM RNN's compositions when the chords within the chord sequences are completely new to

## 6.2. Training of eleven Topologies and Network Compositions

the network or have already been “heard” in the train data. Therefore the train data consists of eight new chord sequences, while four of the chord sequences contain the most frequently used chords in the train data (see table 6.1). The other four chord sequences contain chords that have not been used in the train data. Table 6.2 gives an overview about the 8 chord sequences that have been used as the test data and figure A.1 (see appendix) shows their score.

Chord	C	F	G	Bb	D7	Dm	C7	Em	G7	A7
Occurrences in train data	107	103	72	45	32	30	27	24	21	18

**Table 6.1.:** Most frequently used chords in the train data.

#	Description of test data	Chord denotation
1	Chords occurred in train data	C,F,C,F,C,F,G,G7
2	Chords occurred in train data	Dm,Bb,C,F,Dm,Bb,C7,F
3	Chords occurred in train data	G,G,D7,G,Em,A7,D7,G
4	Chords occurred in train data	C,Dm,Bb,D,C,Bb,D7,G
5	New Chords	Eb6,C7maj,Eb6,Gbm7,Eb6,C7maj,Cdim6,C7maj
6	New Chords	Eb9sus4,Gm7,F#m7,Gm7,D9,Eb9sus4,F#m7,Eb9sus4
7	New Chords	Em9,A13,D9,D9,Em9,A13,D9,D9
8	New Chords	C6maj7,D7,Cm,C#mdim

**Table 6.2.:** Description of the chord denotation of the test data’s chord sequences.

## 6.2. Training of eleven Topologies and Network Compositions

For the experiments that have been done with the LSTM RNN, eleven different network topologies have been trained. Since there don’t exist any heuristics for deciding on an optimal topology when it comes to music composition, different networks have been trained and used for composing melodies. Based on the compositions by the networks, one topology has been selected as the most promising one for fulfilling the compositional goals. The following will detail the training of the networks and the compositions made by the networks.

**Training different Network Topologies** The network topologies that have been trained consist of networks with zero to four hidden layers. Since there doesn’t exist a best-practice for finding an optimal network topology, the number of LSTM Memory Blocks per hidden layer has been chosen by multiples or fractions of twelve, which is the number of input nodes in the input layer. Training took place by starting with a network topology with zero hidden layers, while hidden layers were continuously added. What could be found is

## 6. Experiments

that the melodies started to sound more cohesive and melodically harmonious once more hidden layers were added. This was the case until two hidden layers were added, while the compositions of the network with four hidden layers became more monotonous and were often stuck on one note. For this reason the training has been stopped after having added four hidden layers. Table 6.3 gives an overview about the network topologies that have been trained for the experiments of this thesis.

	#hidden layers	Input Layer #nodes	Hidden layer 1 #MBs*	Hidden layer 2 #MBs*	Hidden layer 3 #MBs*	Hidden layer 4 #MBs*	Output layer #MBs*
1	0	12	n/a	n/a	n/a	n/a	24
2	1	12	3	n/a	n/a	n/a	24
3	1	12	6	n/a	n/a	n/a	24
4	1	12	9	n/a	n/a	n/a	24
5	2	12	3	6	n/a	n/a	24
6	2	12	6	12	n/a	n/a	24
7	2	12	9	18	n/a	n/a	24
8	2	12	12	24	n/a	n/a	24
9	2	12	12	48	n/a	n/a	24
10	3	12	12	24	24	n/a	24
11	4	12	12	12	24	24	24

**Table 6.3.:** Overview about the eleven different network topologies that have been trained.

\*MBs: LSTM Memory Blocks

**Network compositions** Each different network topology described in table 6.3 composed melodies to the eight test chord sequences in figure A.1 (see appendix). As a general tendency it could be noticed that LSTM RNNs with more hidden layers tend to play less notes. The topology with no hidden layer was playing notes more frequently compared to other topologies, while the topology with four hidden layers was playing either almost no notes or was stuck on one note for several bars. Overall, the LSTM RNN is composing melodies that are in key, that is their notes fit harmonically to the given chords. At the same time the network is playing notes noticeably often off beat, that is notes are played a sixteenth or eighth note after one might expect the note to hear. That is why the beginning and ending of the notes of all melodies have been quantized to an eighth note. From eleven topologies, three have been pre-selected and in a second step, after a subjective listening test with three participants, the seventh network topology with two hidden layers and 51 LSTM Memory Blocks in total has been chosen for composing the most pleasant and human-like melodies. The melodies that have been composed by this topology can be seen in figure B.1 (see appendix). These melodies have been used for an evaluation in the form of a listening test, which will be described in the next chapter.

## 7. Evaluation

The previous chapters have explained how the LSTM Recurrent Neural Network has been implemented and the experiments that have been done with the implementation. This chapter will elaborate on the evaluation of the composed melodies in order to assess the quality of the LSTM RNN's compositions. The evaluation has been designed influenced by a framework described in the paper "Towards A Framework for the Evaluation of Machine Compositions", Pearce. The framework is comprised of the following steps:

1. Specifying the compositional aims
2. Inducing a critic from a set of example musical phrases
3. Composing music that satisfies the critic
4. Evaluating specific claims about the compositions in experiments using human subjects

While the compositional aims have already been described, the involvement of a critic is done by creating human compositions as a comparison for the computer compositions. The main focus of the following is the evaluation in experiments using human subjects.

### 7.1. Subjective Listening Test

Due to the nature of the task for this thesis, direct objective measures cannot be applied, since "the evaluation of beauty or aesthetic value in works of art (including music) often comes down to individual subjective opinion", Pearce. Therefore a listening test with human subjects has been designed in order to evaluate the compositional goals for the LSTM RNN:

1. Compose melodies that cannot be distinguished from human melodies.
2. The melodies should sound pleasantly to the listener.

#### 7.1.1. Test Design

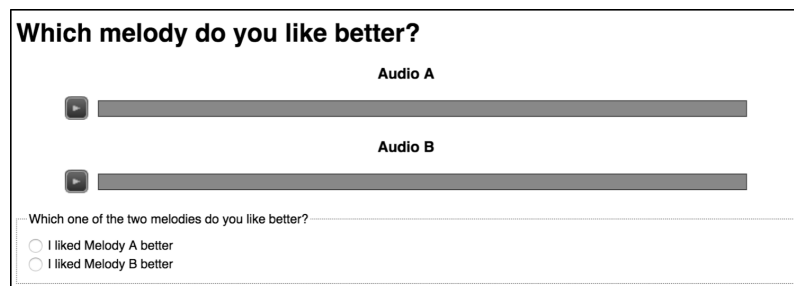
Derived from the compositional goals above, the listening test has been separated into two parts to answer two main questions. The first part aims to answer the question, whether

## 7. Evaluation

the human subjects prefer the computer compositions over human compositions. If the majority of subjects prefers the computer composition, this will indicate that the computer melodies sound more pleasantly to the listener than human melodies. The second part will evaluate the first compositional goal, by letting the subjects classify a melody as computer or human composition. In order to be able to compare computer and human melodies, four musically skilled persons were asked to compose melodies along two of the test chord sequences from figure A.1 (see appendix). The score of the human-composed melodies can be seen in figure C.1 (see appendix). With that, a set of human and computer melodies that have been composed to the same underlying chord sequences is available for running the listening test.

The listening test has been designed as an online test, built with a framework for conducting online listening tests made by the Institute for Data Processing from the Technical University of Munich. The framework was taking care of important features, such as randomization of the questions and making the questions only available for a limited amount of time. This method of conducting an online survey has been chosen to be able to acquire enough participants in a relatively short amount of time. The test has been distributed via several university mailing lists and facebook groups for students and musicians. That way, within five days of running the test 109 participants could be gathered.

**Part One** The first part of the listening test is divided into eight steps, where in each step the subjects are presented a pair of audio files. One file contains the computer melody, the other one the human melody, while the melodies in each step are based on the same underlying chord sequence. For Part One, the subjects have not been told, that some of the melodies have been composed by a computer. At the beginning the participants were instructed to only consider the melody when making a decision and that the chord sequences are not objects of investigation for this survey. In each step the subjects should answer the question “Which melody do you like better”. Since the answer is very dependent on one’s individual taste, the participants were instructed to make their decision freely based on their intuition. An example of one step of the first part is shown in figure 7.1.



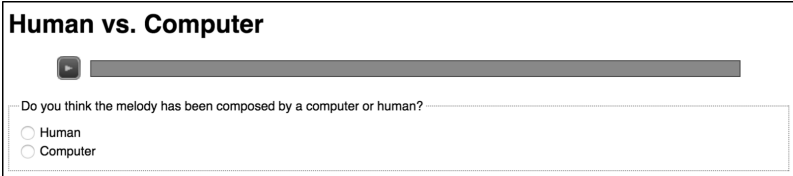
The screenshot shows a web-based interface for a listening test. At the top, the question "Which melody do you like better?" is displayed in bold. Below this, there are two audio player controls. The first is labeled "Audio A" and the second is labeled "Audio B". Each player consists of a play button icon and a progress bar. At the bottom of the interface, there is a text input field with the prompt "Which one of the two melodies do you like better?". Below the input field are two radio button options: "I liked Melody A better" and "I liked Melody B better".

**Figure 7.1.:** One step from the first part of the listening test. Participants were instructed to make their decision freely based on their intuition.



**Part Two** In the second part the human subjects have been told that the melodies from the first part were both composed by a computer and humans. The second part is divided into 16 different steps, where each step contains either one of the eight computer melodies or one of the eight human melodies. The participants should now indicate whether they think the melody has been composed by a human or by the computer. An example for one step from the second part can be seen in figure 7.2. At the end of the second part the subjects were asked to state reasons for making their classification decision. They were asked to indicate whether the following points applied or not:

1. The melody contained wrong sounding notes.
2. There was no coherent progression throughout the melody.
3. I didn't like the melody aesthetically.
4. The rhythm sounded unnatural to me.



The image shows a screenshot of a survey question. At the top, it says "Human vs. Computer". Below this, there is a play button icon and a progress bar. The question is "Do you think the melody has been composed by a computer or human?". There are two radio button options: "Human" and "Computer".

**Figure 7.2.:** One step from the second part of the listening test. Participants should classify a melody as human or computer composition.

**General Information** In order to get an overview about of the human subject group, participants were asked to provide general information about themselves at the end of the listening test. The subjects were asked, whether they play an instrument and for how long, how often they listen to music on average and if music is strongly connected to emotions for them. These are factors that indicate, whether the subject group is musical or not, Muellensiefen (2014).

### 7.1.2. Test Results

The listening test has been conducted for five days with 109 human subjects completing the survey. The quite musical subject group showed a clear preference for human-composed melodies over computer compositions. However, the computer melodies could not easily be detected as computer compositions. The following will detail on the results of the listening test.

## 7. Evaluation

**Subject group** The subject group was quite musical with 64% of the participants playing an instrument for more than three years. Furthermore, almost half of the subject group stated to listen music for two hours a day or more and for more than 90% of the subjects music is strongly connected to emotions, which indicates towards a subject group with a good musical understanding. This is an important aspect to satisfy a qualitative requirement for the evaluation.

Up to 1 year	Up to 3 years	More than 3 years	I do not play an instrument
3.67%	10.09%	64.22%	22.02%

**Table 7.1.:** For how long have you been playing an instrument?

Less than once per week	Every second day	Less than 1h a day	Less than 2h a day	More than 2h day
0.92%	6.42%	12.84%	33.03%	46.79%

**Table 7.2.:** How often do you listen to music on average?

**Results from Part One** In the first part of the listening test the participants were asked whether they prefer the computer melody over the human one, without knowing that computer compositions are amongst the melodies. In about 73% of the cases the human melody was preferred, while still 27% of the participants on average indicated to prefer the computer composition (see table 7.3). This shows a clear preference for human compositions over the computer compositions. This can be proven by calculating a significance test for the null hypothesis: “There is no difference in preference for human or computer compositions”, Dreger (2000). This hypothesis is true if the probability for a preference for human melodies  $p_{HumanMel}$  is equal to 0.5. In this listening test the test result for preferring human melodies is  $\bar{P} = 0.7271$ . The test statistic  $Z_0$  is calculated by equation 7.1, where  $n$  is the number of subjects.

$$Z_0 = \frac{\bar{P} - p_{HumanMel}}{\sqrt{p_{HumanMel}(1 - p_{HumanMel})/n}} \quad (7.1)$$

$$Z_0 = \frac{0.7271 - 0.5}{\sqrt{0.5(1 - 0.5)/109}} = 4.742 \quad (7.2)$$

With a probability of occurrence  $\alpha = 0.01$  the interval for accepting the null hypothesis is  $\bar{K} = [-2.576, 2.576]$ . Since the value for  $Z_0$  is outside of the acceptance region, the null hypothesis is refused with a level of significance of 1%. That means that there is a significant difference in the preference for human compositions over computer compositions.

As described in section 6.1 the first four test chord sequences are familiar to the LSTM RNN, while the last four are unknown to it. Therefore the results have been separated

into the first four and the last four melodies, to notice that there is no big difference in the results between this separation. As for the first four melodies in about 26% of the cases the computer melody was preferred, while this was the case in about 29% of the cases for the last four melodies. Based on these results, it can be claimed that the LSTM RNN performs equally well on familiar and unknown chord sequences.

	Human melodies preferred	Computer melodies preferred
<b>All Testchords</b>	72.71%	27.29%
<b>Testchords 01 to 04</b>	74.08%	25.92%
<b>Testchords 05 to 08</b>	71.33%	28.67%

**Table 7.3.:** The results to the question “Which melody do you like better”. The first row shows the results over all test chord sequences. The second row only for the test chord sequences with familiar chords for the LSTM RNN. The last row only for the unknown test chord sequences.

**Results from Part Two** As for the second part, where the subjects were asked to classify a melody as human or computer composition, the results can be seen in the confusion matrix in table 7.4. Both human and computer melodies have been correctly classified in about 61% of the cases, giving a classification accuracy of 0.6164. This means that there is a distinguishable difference between human and computer compositions, even though this is a not very distinct difference, as 40% of the computer melodies have been falsely classified as human melodies. It has to be noted that the subjects were differently biased on the expectations about a computer composition. Some subjects noted in their feedback, that they expected computer melodies to be “almost perfect”, while others assumed that a computer is generally unable to compose music. Therefore, a possible improvement to this listening test could be to give an example of a computer and a human melody as a reference, before letting the subjects classify the melodies. Further metrics derived from the confusion matrix can be seen in table 7.5.

	Human Melody	Computer Melody
<b>classified as Human Melody</b>	61.81%	38.53%
<b>classified as Computer Melody</b>	38.19%	61.47%

**Table 7.4.:** Confusion matrix for classifying melodies as computer or human composition.

Moreover, the main reason for classifying a melody as computer composition is that the rhythm of the melodies sounds unnatural. The second most prominent reason is that there was no coherent progression throughout the melody (see table 7.6). Multiple reasons could be given for classifying a melody as computer composition, while more than 60% of the subjects stated three or all four reasons.

## 7. Evaluation

	<b>Melodies classified as human</b>	<b>Melodies classified as computer</b>
<b>Precision</b>	0.6160	0.6168
<b>Recall</b>	0.6181	0.6147
<b>F-Score</b>	0.6170	0.6157

**Table 7.5.:** The metrics Precision, Recall and F-Score calculated for the melodies classified as human and computer composition.

	<b>The melody contained wrong sounding notes</b>	<b>I did not like the melody aesthetically</b>	<b>There was no coherent progression throughout the melody</b>	<b>The rhythm sounded unnatural to me</b>
<b>Votes for "Applies"</b>	53.21%	62.39%	71.56%	75.23%

**Table 7.6.:** The reasons given for classifying a melody as a computer composition.

**Conclusions from the results** The questions that should be answered through the listening test and that will eventually evaluate the compositional goals of the LSTM RNN are:

1. Are melodies composed by a computer perceived to sound pleasantly?
2. Are melodies composed by a computer being recognized as such?

For the first question, the results show that there is a significant preference for human composed melodies. However, in 27% of the cases the computer melody was preferred over the human one, which indicates that some of the composed melodies were in fact sounding pleasantly to the listener. In that regard one could claim that the LSTM RNN partly composes melodies, that sound pleasantly to the listener.

As for the second question, the computer melodies were correctly classified in about 60% of the cases, while in 40% of the cases they were misclassified. This means that the computer melodies contain elements, which make them recognizable as such. However there does not seem to be a decisive computer-like sounding element in the melodies, as the gap between correct and false classifications is relatively small. In regard of the first compositional goal, the LSTM RNN failed to compose melodies that cannot be distinguished from human ones, on the basis of the listening test's results. Still, this goal would require further evaluation in another listening test, where the subjects are not told, that the melodies have been composed by a computer.

## 8. Conclusion

This thesis described a method to compose melodies with LSTM Recurrent Neural Networks. After giving an overview about the state of the art and an introduction to Neural Networks, the implementation of the algorithm was described. The experiments that have been done with the implementation were discussed and finally the computer compositions have been evaluated using human subjects in a listening test.

The main goal for this thesis was to implement a long-short term memory Recurrent Neural Network, that composes melodies that sound pleasantly to the listener and cannot be distinguished from human melodies. As the evaluation of the computer compositions has shown, the LSTM RNN composed melodies that partly sounded pleasantly to the listener. However, in the majority of the cases the human compositions were preferred, thus the LSTM RNN did not fully succeed in fulfilling the first requirement. In addition, the computer compositions could be recognized as such in about 60% of the cases, making them distinguishable from human compositions. Still, in 40% of the cases the computer melodies were misclassified as human melodies. Overall the LSTM RNN did not succeed in completely fulfilling the main goals for this thesis, but it did in fact compose very interesting and nice sounding melodies at parts, that satisfy those goals.

Further improvements can be applied to the algorithm in order to satisfy the requirements. For example, more train data could be used as this is very decisive for the quality of the LSTM RNNs compositions. On top, by defining the compositional goals more specifically, the train data can be improved by tailoring it to those goals. The goal of this thesis for the algorithm was to compose melodies of any kind, however the compositional goals could be specified to compose melodies within a certain genre, in a certain tempo, etc. Moreover, a different way of representing music apart from the piano roll representation could improve the algorithm, by letting more versatile musical expressions influence the composition process.

A final result is an implementation of a LSTM Recurrent Neural Network, that composes a melody to a given chord sequence, which, apart from any requirements of this thesis, can be used as a creative tool of inspiration for composers and music producers.



# Appendices





# A. Test Data Score

The image displays eight numbered musical staves, each containing a sequence of chords. The notation is in bass clef with a 4/4 time signature. The chords are represented by groups of notes on the staff, with some notes marked with an 'x' to indicate they are not to be played. The sequences are as follows:

- 1: C major triad (C, E, G) in four positions.
- 2: C major triad (C, E, G) in four positions.
- 3: C major triad (C, E, G) in four positions.
- 4: C major triad (C, E, G) in four positions.
- 5: C major triad (C, E, G) in four positions.
- 6: C major triad (C, E, G) in four positions.
- 7: C major triad (C, E, G) in four positions.
- 8: C major triad (C, E, G) in four positions.

Figure A.1.: The score of the 8 chord sequences that have been used as the test data.



## B. Network Compositions Score

The figure displays eight numbered musical compositions, each consisting of two staves (treble and bass clef).  
1. Treble clef, 4/4 time, key of D major. Melody: D4 quarter, E4 quarter, F#4 quarter, G4 quarter, A4 quarter, B4 quarter, C5 quarter, B4 quarter, A4 quarter, G4 quarter, F#4 quarter, E4 quarter, D4 quarter.  
2. Treble clef, 4/4 time, key of B-flat major. Melody: Bb4 quarter, Ab4 quarter, G4 quarter, F4 quarter, E4 quarter, D4 quarter, C4 quarter, Bb3 quarter, Ab3 quarter, G3 quarter, F3 quarter, E3 quarter, D3 quarter.  
3. Treble clef, 4/4 time, key of D major. Melody: D4 quarter, E4 quarter, F#4 quarter, G4 quarter, A4 quarter, B4 quarter, C5 quarter, B4 quarter, A4 quarter, G4 quarter, F#4 quarter, E4 quarter, D4 quarter.  
4. Treble clef, 4/4 time, key of B-flat major. Melody: Bb4 quarter, Ab4 quarter, G4 quarter, F4 quarter, E4 quarter, D4 quarter, C4 quarter, Bb3 quarter, Ab3 quarter, G3 quarter, F3 quarter, E3 quarter, D3 quarter.  
5. Treble clef, 4/4 time, key of D major. Melody: D4 quarter, E4 quarter, F#4 quarter, G4 quarter, A4 quarter, B4 quarter, C5 quarter, B4 quarter, A4 quarter, G4 quarter, F#4 quarter, E4 quarter, D4 quarter.  
6. Treble clef, 4/4 time, key of B-flat major. Melody: Bb4 quarter, Ab4 quarter, G4 quarter, F4 quarter, E4 quarter, D4 quarter, C4 quarter, Bb3 quarter, Ab3 quarter, G3 quarter, F3 quarter, E3 quarter, D3 quarter.  
7. Treble clef, 4/4 time, key of D major. Melody: D4 quarter, E4 quarter, F#4 quarter, G4 quarter, A4 quarter, B4 quarter, C5 quarter, B4 quarter, A4 quarter, G4 quarter, F#4 quarter, E4 quarter, D4 quarter.  
8. Treble clef, 4/4 time, key of B-flat major. Melody: Bb4 quarter, Ab4 quarter, G4 quarter, F4 quarter, E4 quarter, D4 quarter, C4 quarter, Bb3 quarter, Ab3 quarter, G3 quarter, F3 quarter, E3 quarter, D3 quarter.

**Figure B.1.:** The melodies composed by the LSTM RNN that have been used for the listening test. The melodies were composed to the chords from figure A.1.



## C. Human Melodies Score

The image displays a musical score for human melodies, organized into eight systems. Each system is numbered 1 through 8 on the left. System 1 consists of two staves: the top staff is in treble clef with a key signature of one flat and a 4/4 time signature, and the bottom staff is in treble clef. System 2 consists of one staff in treble clef. System 3 consists of two staves in bass clef with a key signature of two sharps. System 4 consists of two staves in bass clef. System 5 consists of two staves in treble clef. System 6 consists of two staves in bass clef. System 7 consists of two staves in treble clef with a key signature of two sharps. System 8 consists of two staves in treble clef. The score includes various musical notations such as notes, rests, and clefs.

**Figure C.1.:** The score of the human melodies, which have been used for the listening test as a comparison to the computer compositions.



# Bibliography

- Wikipedia: Musical instrument digital interface. 2015. URL [https://de.wikipedia.org/wiki/Musical\\_Instrument\\_Digital\\_Interface](https://de.wikipedia.org/wiki/Musical_Instrument_Digital_Interface)(lastaccessed:04.11.2015).
- L. Bottou. Large-scale machine learning with stochastic gradient descent. In *COMP-STAT'2010*. 2010.
- R. Dean. *The Oxford Handbook of Computer Music*. Oxford University Press, 2009.
- E.K. Dreger. *Statistik*. Gabler, 2000.
- J. Eck, Douglas; Schmidhuber. *A First Look at Music Composition using LSTM Recurrent Neural Networks*. Technical Report, IDSIA, 2002.
- F. Gers. Long short-term memory in recurrent neural networks. In *Unpublished PhD dissertation, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland*, 2001.
- S. Haykin. A comprehensive foundation. In *Neural Networks*, 2(2004), 2004.
- L. Hiller, Lejaren; Isaacson. *Experimental Music - Composition with an electronic computer*. MCGRAW-HILL BOOK COMPANY, 1959.
- D. Johnson. Composing music with recurrent neural networks. August 2015. URL <http://www.hexahedria.com/2015/08/03/composing-music-with-recurrent-neural-networks/>(lastaccessed: 26.10.2015).
- D.J. Levitin. *Der Musik Instinkt*. Spektrum Akademischer Verlag, 2006.
- J.E.C. Lipton, Zachary; Berkowitz. In A critical review of recurrent neural networks for sequence learning, 2015.
- M. Mozer. Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multiscale processing. In *Connection Science*, 1994.
- B.M.J.S.L. Muellensiefen, Daniel; Gingras. The musicality of non-musicians: An index for assessing musical sophistication in the general population. In *PLoS ONE 9 e89642. doi 10.1371/journal.pone.0089642*, 2014.
- A. Ng. Lecture notes, sparse autoencoder. 2012a.

## Bibliography

- A. Ng. Lecture notes, supervised learning. 2012b.
- G. Nierhaus. *Algorithmic Composition - Paradigms of Automated Music Generation*. SpringerWienNewYork, 2009.
- G. Papadopoulos, George; Wiggins. Ai methods for algorithmic composition: A survey, a critical view and future prospects. In *School of Artificial Intelligence, Division of Informatics, University of Edinburg*.
- G. Pearce, Marcus; Wiggins. Towards a framework for the evaluation of machine compositions.
- N.L.C. Principe, Jose; Euliano. *Neural and Adaptive Systems: Fundamentals Through Simulation*. 1997.
- R. Rojas. *Neural Networks, A Systematic Introduction*. Springer, 1996.
- G.W.R. Rumelhart, David; Hinton. Learning representations by back-propagating errors. In *Nature*, 1986.
- P. da Silva. David cope and experiments in musical intelligence. 2003.
- M. Simoni. *Algorithmic Composition: A Gentle Introduction to Music Composition Using Common LISP and Common Music*. Ann Arbor, University of Michigan Library, <http://dx.doi.org/10.3998/spobooks.bbv9810.0001.001>, 2003.
- J. Stange-Elbe. *Computer und Musik: Grundlagen, Technologien und Produktionsumgebungen der digitalen Musik*. De Gruyter Oldenbourg, 2015.
- V.P.J. Svozil, Daniel; Kvasnicka. Introduction to multi-layer feed-forward neural networks. In *Elsevier*, 1997.
- F.F. Wilson. Music and mathematics - from pythagoras to fractals. In *Oxford University Press*, 2003.
- S. Windisch. Mathematik zum anfassen - musikalisches wuerfelspiel.
- H. Zwicker, Eberhard; Fastl. *Psychoacoustics Facts and Models*. Springer, 1999.