JACOBS
UNIVERSITY

# Art in Echo State Networks: Music Generation

by

**Aulon Kuqi**

Bachelor Thesis in Computer Science

Prof.Dr. Herbert Jaeger
Name and title of the supervisor

Date of Submission: May 24, 2017

Jacobs University — School of Engineering and Science

With my signature, I certify that this thesis has been written by me using only the indicates resources and materials. Where I have presented data and results, the data and results are complete, genuine, and have been obtained by me unless otherwise acknowledged; where my results derive from computer programs, these computer programs have been written by me unless otherwise acknowledged. I further confirm that this thesis has not been submitted, either in part or as a whole, for any other academic degree at this or another institution.

Signature Place, Date

## Abstract

The ability to construct a model of music from examples presents a great intellectual challenge that, if successfully met, could foster a range of new creative applications. This guided thesis proposal describes a music generation task using Echo State Networks (ESNs); a research based on Tomas Pllaha's Bachelor thesis. The task is as follows: feeding multi-track 'melodies' into the system should result in a fitting bass track for those melodies.

The usage of such a system falls into four potential categories: practice, improvisation, composing, and further research on creative AI. The generated bass track adds a further musical dimension while practicing scales and exercises; it helps composers that lack knowledge of the instrument complete their music; it makes improvisation sessions more lively, especially when the bass is missing or needed; it forwards research on creative AI.

Recurrent Neural Networks (RNNs) and other Machine Learning techniques have been previously used for music generation tasks. The novelty in this approach lies in using ESNs. ESNs are generally faster and much easier to train than other classical RNNs. They are very suitable for prediction of time series (data points - measured typically in equally-spaced time intervals). Music, as a sequence of tones measured at different points in time, fits perfectly with ESNs.

This guided research will focus on computational processing of a suitable track while listening to pre-recorded leading tracks. For this purpose, MIDI files will be used as training data and output.

## Contents

# 1 Introduction

The ability to construct a model of a musical style from examples presents a great intellectual challenge that, if successfully met, could foster a range of new creative applications. The act of musical composition involves a highly structured mental process. Although it is complex and difficult to formalize, it is clearly far from being a random activity [1].

Researchers commonly agree that expectations based on recent-past context guide musical perception [1]. A generative theory of music can be constructed by explicitly coding music rules in some logic or formal grammar. This approach is called an *expert system* and although it can achieve impressive results, it requires extensive exploitation of musical knowledge often specific to a composer or style [2].

Several methods have been explored to tackle music generation tasks. Markov chains have been exploited for music composition ([15]), however, they suffer from drawbacks. The most significant drawback is that the structure of music violates the Markov hypothesis: the output at each step *only* depends on the output of the previous step. Recurrent Neural Networks (RNNs) have also been used for music generation tasks. One example is CONCERT ([3]), a network that composes melodies with harmonic accompaniment. This network achieved satisfying results in 1994, however, it had a complex structure.

A more recent example is WaveNet ([4]). Wavenet is based on Convolutional Neural Networks, the deep learning technique that has been working very well in image classification and generation tasks, in the past few years. Its most promising purpose is to enhance text-to-speech applications by generating a more natural flow in vocal sound. However, its method can also be applied to music as both the input and output consists of raw audio. It uses raw audio as input. Therefore it can generate any kind of instrument, and even any kind of sound. However, the algorithm is computationally expensive. It takes minutes to train on a second of sound. Furthermore, WaveNet's sample outputs only contain 10 seconds of generated piano notes.

Focusing on RNNs extensive use and their satisfying results for music generation tasks, a type of Recurrent Neural Network (RNN) was selected for this thesis: Echo State Networks (ESNs) (Figure 1). ESNs contain a set of input neurons, a set of output neurons, and a reservoir of randomly-generated recurrent neurons. This reservoir creates the ability to approximate arbitrary dynamical systems. Unlike other RNNs, where all weights need to be updated, in ESNs only the output weights need to be learned, fixing reservoir weights with a random value. The lack of cyclic dependencies between the trained readout connections makes ESN training a simple linear regression task [5]. Ridge regression yields more stable solutions [6].

RNNs capitalize on a massive short-term memory, and can be used for different tasks, including: time series prediction and event classification. The advantages of ESNs are the following:

1. ESNs are easy to train and solutions are calculated in a single step.
2. Their Echo State Property [7] makes them very suitable for this particular task. Events in the more recent past affect the output at the present step more than events that occurred earlier.

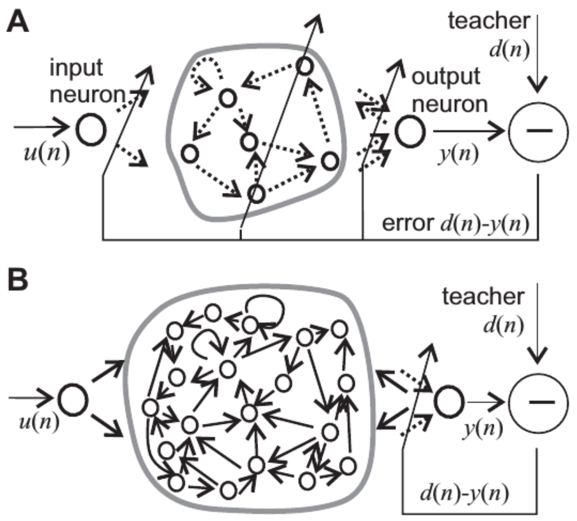

Figure 1: **(A)** Schema of other approaches to RNN learning. **(B)** Schema of ESN approach. Solid bold arrows: fixed synaptic connections, dotted arrows: adjustable connections. Both approaches aim at minimizing the error $\mathbf{d}(n) - \mathbf{y}(n)$, where $\mathbf{y}(n)$ is the network output, and $\mathbf{d}(n)$ is the "teacher" time series observed from the target system (figure and text taken from [5])

Section $2$ contains the statement and motivation of research, along with a detailed explanation on ESN. Section $3$ describes representation of musical data. Section $4$ describes the network setup. Section $5$ contains conducted experiments. Section $6$ describes the evaluation criteria. Section $7$ concludes with a summary of the main findings of this thesis and a list of known drawbacks, and discusses potential focus areas of future research on this topic.

## 2 Statement and Motivation of Research

Inspired from biological neural networks, artificial neural networks have units analogous to neurons, and connections that carry 'impulses' between units, analogous to synapses and axons. This biological inspiration can be used to exploit simulated creativity from artificial neural networks. Thus, tasks requiring 'creativity', such as music generation, fit very well in the RNN model. It is important to clarify that the term 'creativity' is used in a loose sense. The drawback using RNN is that they require significant computational power and an advanced level of expertise to build such a system. This is where ESNs come in hand, with their low computational power required, and their relatively simple architecture. The trade-off for this gain in speed is the increased reservoir size.

Subsection $2.1$ gives a formal description of ESNs. Subsection $2.2$ is the statement of Research, and subsection $2.3$ includes some research questions to be addressed during experimentation.

## 2.1 Echo State Networks

ESNs present an alternative that often works well enough even without full adaptation of all network weights [8]. They are conceptually simple and computationally inexpensive. However, this apparent simplicity of ESNs can be deceptive.

ESNs are applied to supervised temporal ML tasks where for a given input signal $\mathbf{u}(n)$, a desired target output signal $\mathbf{y}^{target}(n)$ is known. The task is to learn a model with output $\mathbf{y}(n)$, where $\mathbf{y}(n)$ matches $\mathbf{y}^{target}(n)$ as well as possible, minimizing an error measure $E(\mathbf{y}(n), \mathbf{y}(n))$, and, more importantly, generalize well to unseen data. The error measure $E$ is typically a Mean-Square Error (MSE). ESNs are fed an input vector $\mathbf{u}$ of size $K$,

$$\mathbf{u}(n) = (u_1(n) \ldots u_K(n))$$

a reservoir (random vector) $\mathbf{x}$ of size $N$,

$$\mathbf{x}(n) = (x_1(n) \ldots x_N(n))$$

and an output vector $\mathbf{y}$ of size $L$.

$$\mathbf{y}(n) = (y_1(n) \ldots y_L(n))$$

The input-to-reservoir connection weights are stored in a matrix $\mathbf{W}^{in}$ of size $N \times K$. The reservoir connection weights are stored in a matrix $\mathbf{W}$ with size $N \times N$ and the output connection weights are stored in a matrix $\mathbf{W}^{out}$ with size $L \times K + N$.

$$\mathbf{W}^{in} = (w_{ij}^{in}) \;\; \mathbf{W} = (w_{ij}) \;\; \mathbf{W}^{out} = (w_{ij}^{out})$$

The output units may optionally project back to internal units with connections whose weights are collected in a $N \times L$ backprojection weight matrix $\mathbf{W}^{fb}$

$$\mathbf{W}^{fb} = (w_{ij}^{fb})$$

A weight value of ’zero’ can be interpreted as ”no connection”. Output units may have connections not only from internal units, but also from input units and (rarely) from output units.

In addition, let $f$ be a sigmoid function (either the logistic function, or *tanh*). The network is then governed by the following state update equation:

$$\mathbf{x}(n+1) = f(\mathbf{W}^{in}\mathbf{u}(n+1) + \mathbf{W}\mathbf{x}(n) + \mathbf{W}^{fb}\mathbf{y}(n)) \tag{1}$$

where $f$ denotes the component-wise application of the activation function. ESNs use an RNN type with leaky-integrated discrete-time continuous-value unit. The update equation is:

$$\mathbf{x}(n+1) = (1-\alpha)\mathbf{x}(n) + \alpha f(\mathbf{W}^{in}\mathbf{u}(n+1) + \mathbf{W}\mathbf{x}(n) + \mathbf{W}^{fb}\mathbf{y}(n)) \tag{2}$$

where $\alpha \in (0, 1]$ is the leaking rate. The network is then exploited on testing data by producing an output:

$$\mathbf{y}(n) = g(\mathbf{W}^{out}\mathbf{x}(n)) \tag{3}$$

Where $g$ is the output activation function (typically a sigmoid or identity function).

The learning task is computing the output weights $W^{out}$ by minimizing the mean squared error. This can be achieved by linear regression; ridge regression, however, is the most frequently used method, due to its regularization power [6]. In cases where the ESN is trained on a signal generation task with output feedback (such as here), ridge regression provides and enhanced stability. The desired outputs for all timesteps are stored in a matrix $\mathbf{D}_{n_{max} \times L}$, where $n_{max}$ is the length of the training sequence. The extended reservoir states $[\mathbf{x}; \mathbf{u}]$ are stored in a matrix $\mathbf{S}_{n_{max} \times (N+K)}$.

Ridge regression (or Tikhonov regularization): $\mathbf{W}^{out} = (\mathbf{R} + \gamma^2 I)^{-1} P$, where $\mathbf{R} = \mathbf{S}'\mathbf{S}$ is the correlation matrix of the extended reservoir state, $\mathbf{P} = \mathbf{S}'\mathbf{D}$ is the cross correlation matrix of the extended network states and the desired outputs, $\gamma$ is a regularization parameter, and $\mathbf{I}$ is the identity matrix. Other solutions include the Wiener-Hopf solution: $\mathbf{W}^{out} = \mathbf{R}^{+}\mathbf{P}$.

The algorithm for training an ESN can be summarized as follows:

**Given:** Input training signal $\mathbf{u}(n) \in \mathbb{R}^K$ and a desired output signal $\mathbf{y}_{teacher}(n) \in \mathbb{R}^L$, where $n \in \{1, 2, \ldots, T\}$ is the discrete time step and $T$ represents the number of training data points.

**Wanted:** A trained ESN that generalizes well to new data, characterized by the weight matrices $\mathbf{W}^{in}, \mathbf{W}, \mathbf{W}^{out}, \mathbf{W}^{fb}$, whose output $\mathbf{y}(n)$ approximates teacher signal $\mathbf{y}_{teacher}(n)$.

**Algorithm[7]:**

1. Procure an untrained DR network $(\mathbf{W}^{in}, \mathbf{W}, \mathbf{W}^{fb})$ which has the *echo state property*, and whose internal units exhibit mutually interestingly different dynamics when excited. This involves some sub-steps:

   (a) Randomly generate an internal weight matrix $\mathbf{W}_0$.

   (b) Normalize $\mathbf{W}_0$ to a matrix $\mathbf{W}_1$ with unit spectral radius by putting $\mathbf{W}_1 = \frac{1}{|\lambda_{max}|}\mathbf{W}_0$, where $|\lambda_{max}|$ is the spectral radius of $\mathbf{W}_0$.

   (c) Scale $\mathbf{W}_1$ to $\mathbf{W} = \alpha \mathbf{W}_1$, where $\alpha < 1$, whereby $\mathbf{W}$ obtains a spectral radius of $\alpha$.

   (d) Randomly generate input weights $\mathbf{W}^{in}$ and output backpropagation weights $\mathbf{W}^{fb}$.

   Then the untrained network $(\mathbf{W}^{in}, \mathbf{W}, \mathbf{W}^{fb})$ is an Echo State Network, regardless of how $\mathbf{W}^{in}$, $\mathbf{W}^{fb}$ are chosen.

2. Train the network.

   (a) Initialize the activations of the internal neurons to an arbitrary value, e.g. $\mathbf{x}(0) = \mathbf{0}$.

   (b) Drive the network using the training data for times $n = 0 \ldots T$, by presenting the input $\mathbf{u}(n)$, and by teacher-forcing the teacher output $\mathbf{y}_{teacher}(n-1)$, computing

   $$\mathbf{x}(n) = (1-\alpha)\mathbf{x}(n-1) + \alpha f(\mathbf{W}^{in}\mathbf{u}(n) + \mathbf{W}\mathbf{x}(n-1) + \mathbf{W}^{fb}\mathbf{y}_{teacher}(n-1))$$

   At time $n = 0$, where $\mathbf{y}_{teacher}(n-1)$ is not defined, use $\mathbf{y}_{teacher}(n-1) = 0$. Collect all $\mathbf{x}(n)$, where $n > washout\_time$ into a matrix $\mathbf{X}$. Discard $\mathbf{x}(n)$, where

$n < washout\_time$. $Washout\_time$ is a time $T_0 > 1$ where initial transient dynamics of the network which are invoked by the arbitrary network starting state, are washed out. Collect all $\mathbf{y}_{teacher}(n)$ into a matrix $\mathbf{D}$.

3. Compute the linear readout weights $\mathbf{W}^{out}$ from the reservoir by minimizing the Mean Squared Error (MSE) between $\mathbf{y}(n)$ and $\mathbf{y}_{target}(n)$, using ridge regression
   (a) $\mathbf{R} = \mathbf{X}'\mathbf{X}$
   (b) $\mathbf{P} = \mathbf{X}'\mathbf{D}$
   (c) $\mathbf{W}^{out} = (\mathbf{R} + \gamma^2 I)^{-1}\mathbf{P}$
4. Exploitation. The network $(\mathbf{W}^{in}, \mathbf{W}, \mathbf{W}^{out}, \mathbf{W}^{fb})$ is ready for use. It can be driven by novel input sequences $\mathbf{u}(n)$ using
$$\mathbf{x}(n) = (1-\alpha)\mathbf{x}(n-1) + \alpha f(\mathbf{W}^{in}\mathbf{u}(n) + \mathbf{W}\mathbf{x}(n-1) + \mathbf{W}^{fb}\mathbf{y}(n-1))$$
$$\mathbf{y}(n) = f(\mathbf{W}^{out}\mathbf{x}(n))$$
   It is very important to note that $\mathbf{y}(n)$ here is *not* $\mathbf{y}_{teacher}(n)$.

This algorithm generates a reservoir which ensures the Echo State Property. This property is required in order to be able to approximate the desired output signal. That is, the current network state (activation values of internal neurons) is uniquely determined by the history of the input and the teacher-forced output, after the network has been run longer than its washout time (i.e the dynamical reservoir asymptotically forgets its initial state).

## 2.2 Statement of Research

This guided research is built upon Tomas Pllaha's Bachelor thesis([9]). The pitch range was suited for the bass guitar, and more training data was used. Tomas' network was re-implemented with more 'generic' songs that are closer to a genre-specific musical-progression, and less musically complex. 'Generic' in this case means a song whose bass-melody is repetitive, yet not monotonous. Examples include music of the genres of rock n' roll, hard rock, classical rock, blues, etc. Additionally, the network is upgraded to support two-note guitar chords from the lead track. Although, optimization for such an extension was not achieved.

Restating Tomas Pllaha's statement of research: "Through this guided research, the suitability of ESNs for the music accompaniment task will be tested. Additionally, an attempt will be made to find the network parameters that yield the best performance, by manual experimentation. Ideally, the presentation of ESNs will produce faster and more 'creative' solutions than other RNNs."

One of ESNs advantages is the fading short-term memory. The Echo State Property uniquely determines the current network state by the history of the input and teacher-forced output after the network has been run for a long time. The effect of the initial state of the reservoir on the output vanishes after sufficiently long runs, thus the output is a function of the input vectors only. In other words, the recent input affects the output more than earlier input. This property is very suitable for music generation tasks, because songs may change as they progress, but close sequences have to fit into the same melody. This short-term memory ([7]) property of ESNs is expected to be of great use for the music generation task.

However, further challenges are presented by the instrumental accompaniment task. One restriction is creating a bassline without any leading tracks. Other restrictions can be easily handled. For example, the rhythm and duration of a musical measure will not exceed $\frac{4}{4}$.

This research started by re-implementing Tomas' network. Then the data was adjusted to better fit notes on an electric bass. Afterwards, the network was trained with more generic songs. In the end, the network leading track input was upgraded to support simple two-note chords.

### 2.3 Research Questions

Through this guided research the following questions will be addressed:

1. How can ESNs be used as an instrumental accompaniment system?
2. What network parameters yield best results?
3. Will the network produce aesthetically-pleasing, or even new, music?
4. How does the performance of the system depend on the input?

## 3 Representation of Musical Data

As mentioned in the introduction, a generative theory of music can be constructed by explicitly coding music rules in some logic or formal grammar. However, this requires extended expertise. Thus, it is of crucial importance *how* musical notes are represented and *what* information is retrieved from them. More knowledge in the system equals better prediction, and more information from a single note equals a more complex network. MIDI carries event messages that specify notation, pitch and velocity, control signals for parameters such as volume, vibrato, audio panning, cues, and clock signals that set and synchronize tempo between multiple devices. A balanced trade-off will be achieved by representing notes with two main features: pitch and duration. All songs in a major key are transposed to $G_{major}$, and respectively all songs in a minor key are transposed to $E_{minor}$ ($G_{major}$'s relative minor). The representation follows Tomas' model ([9]).

Subsections $3.1$, $3.2$, $3.3$, describe the representation of pitch, duration, and rhythm, respectively. Subsection $3.4$ describes the procedures of selecting and manipulating the training data.

### 3.1 Representation of Pitch

The pitch of a note is its frequency. A 24-fret, 5-string bass guitar produces notes from $B0$ to $G5$. Thus, notes range from $G0$ to $G5$, encircling all notes produced by the bass guitar. Whenever chords appear on the training set, only the lowest note are used. They are represented by CONCERT ([3]).

The pitch of each note is encoded as a 5-dimensional vector. In this vector, the first unit has a value proportional to the logarithm of the absolute pitch of the note. The second and third units are $(x, y)$ coordinates of the position of the note in the chroma circle. Similarly, the fourth and fifth units code the $(x, y)$ coordinates of the note in the circle of fifths. This representation is based on Shepard's theory of generalization ([10]), according to which the perceived similarity of two items decreases exponentially with the distance between them in an internal or psychological representational space ([3]).

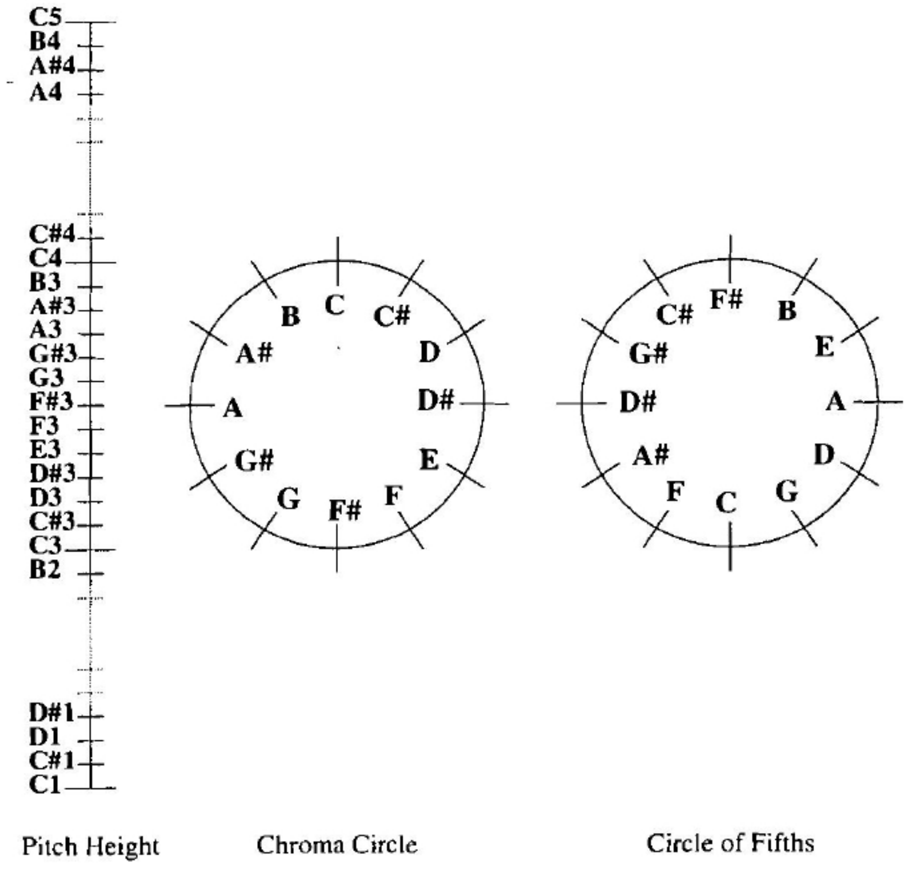

Figure 2: Shepard's (1982) pitch representation [3]

The logarithmic transformation of absolute pitch places tonal half-steps at equal distances from one another along the pitch axis. In the chroma circle, neighboring pitches are a tonal half-step apart. In the circle of fifths, the perfect fifth of a pitch is the pitch that follows immediately counterclockwise ([3]).

The relative importance of each of these three components in determining the similarity of two different notes can be adjusted by changing the diameters of the chroma circle and the circle of fifths. Both circles are be given same diameter, making it equal to the length of one octave on the pitch axis ([11]).

## 3.2 Representation of Duration

Duration is represented by an integer value $\frac{1}{16} \rightarrow \frac{16}{16}$. The duration of each note is encoded with only one duration height dimension. The duration height is proportional to the logarithm of the duration. This logarithmic transformation follows the general psychophysical law (Fechner's law) relating stimulus intensity to perceived sensation ([3]).

| Note | Distance from C1 |
|---|---|
| C1# | 2.0069 |
| D1 | 1.4530 |
| D1# | 2.0616 |
| E1 | 2.5386 |
| F1 | 2.1667 |
| F1# | 3.0000 |
| G1 | 2.3154 |
| G1# | 2.7889 |
| A1 | 2.5000 |
| A1# | 2.1858 |
| B1 | 2.7131 |
| C2 | 2.0000 |
| C2# | 2.9486 |
| D2 | 2.7285 |
| D2# | 3.2016 |
| E2 | 3.6209 |
| F2 | 3.4681 |
| F2# | 4.1231 |
| G2 | 3.7454 |
| G2# | 4.1366 |
| A2 | 4.0311 |
| A2# | 3.9299 |
| B2 | 4.3237 |
| C3 | 4.0000 |

Figure 3: Euclidean Norms of differences between notes

## 3.3 Representation of Rhythm

The rhythm is taken from percussion instruments of the training and testing set and is encoded with a single unit. This unit has a value of 1 whenever a percussion instrument is hit, and 0 at any other time.

## 3.4 Selecting and Manipulating the Training Data

A personal collection of *Guitar Pro* files is used. *Guitar Pro* is a software that allows you to write/read/listen musical sheets, as well as import/export other musical file formats (such as MIDI). The collection contains approximately $60'000$ songs, grouped by artist (or band). Unfortunately, due to time constraints, lack of automation, and the quality of each file, only a fraction of this large dataset is used. The dataset used for experimentation contains $100$ training songs, in addition a 'secret' set of $10$ testing songs is available for after the network is optimized. Such *Guitar Pro* files can be found online for free on different websites (UltimateGuitar, etc.). The conversion from the original dataset onto the dataset used in Matlab follows these steps:

1. Manually select artists (or bands) that contain music which fits the *generic bass-line* category mentioned above. All these files are in a *Guitar Pro* format.
2. For every artist (or band), open every song available using the *Guitar Pro* software. Since these *Guitar Pro* files are not written by the same person, and since these files are not original musical sheets from the artists themselves, manually check if these two conditions are met:
   (a) The *Guitar Pro* file contains *at least* these three tracks: a bass track, a guitar track, a drum track.
   (b) The *Guitar Pro* file does include the full song.

   Delete the files that do not meet the two requirements.
3. After $Step\ 2$, a smaller dataset is present. For every artist (or band), open every song using the *Guitar Pro* software. Use a feature of the software to transpose *every* track from the original key to the desired key of $G_{major}$ (or its relative minor, $E_{minor}$). Depending on the original key of the song, this leads to all notes being shifted up or down the pitch scale. Caution must be taken, for a low enough shift might put bass notes outside of the electric-bass pitch range, thus deleting the note from the file, and ultimately losing the note. Conversely, a high enough shift might put bass notes on a higher pitch, thus resembling guitar notes. This is not desired. However, the most problematic part is finding the correct key of a song. Unfortunately this is not a trivial task, and there is no explicit information in the file regarding the key of the song. In order to find the correct key, cross-referencing combined with a personal expertise is applied.
4. Export to MIDI. For every song, after transposing all tracks to the desired key, mute all tracks **except** one guitar track, the bass track, and the drums track. The network only supports three tracks at the moment. Muting these tracks will exclude them from being exported into the MIDI format. Also it is important to note that the tracks should be in the following order: guitar - bass - drums. This is easily achieved by changing the order of tracks in the software.
5. Select songs to be used for training, and songs to be used for testing. Since the songs were grouped by artist, represent the testing set with songs by including various artists from the training set.
6. Collect all training MIDI files into one MIDI file. This is done using a DAW (Digital Audio Workstation), in this case *Cubase*. Caution must be taken when importing multiple MIDI files because some information might be lost. This is due to default settings on a DAW. It is highly important to import information regarding *tempo* and *time signature* from MIDI, especially while dealing with a large set of MIDI files. Since each MIDI import will create 3 additional (separate) tracks, manual relocation is required. All guitar parts tracks all MIDI files **must** be on one track; all bass tracks from all MIDI files **must** be on another track; all drum tracks from all MIDI files **must** be on a third track. This leaves you with only three tracks which contain information about all the training set. In this case, the training is a 7 hour 15 min MIDI file. Additionally the DAW generated a *tempo track* which contains information about all tempo and time signature changes throughout the training set.
7. Export a singe training MIDI file, containing all the information.

8. On a Linux machine, use the tools *csvmidi* and *midicsv* to convert the MIDI to a CSV (comma-separated value) file. The CSV is chosen because it is easier and faster to read onto Matlab. If a language that supports MIDI input is chosen, this step is disregarded. However, the conversion $MIDI \rightarrow CSV$ is lossless, thus either choice is allowed.

All training data is now in a CSV format. This format is read onto Matlab.

## 4 Network Setup

First and foremost, a replica of Tomas' network was built, with the changes mentioned in the previous section. The network was trained from scratch with a larger data set of less complex music. As mentioned earlier, songs are selected from the genres of rock n' roll, hard rock, classical rock, and blues.

Subsection $4.1$ explains how the network was designed. Subsection $4.2$ explains how network parameters were optimized. Subsection $4.3$ explains how the network was trained.

### 4.1 Network Design

The CSV file containing all training data is further divided and simplified in Matlab:

1. Use a Matlab function that reads a CSV file into a matrix (cell-array), called *data*.
2. Divide the matrix containing all data into three different matrices: guitar, bass, and drums.
3. To create the guitar information matrix, read the guitar track from *data*, and take only one note at a time, with its duration. This means from chords, only the first note is selected. A second implementation which reads two notes from a chord will be explained in Section $5$.
4. To crate the bass information matrix, read the bass track from *data*, take only one note at a time with its duration. Bass tracks do not usually have chords (or harmonic intervals) thus no information is lost here.
5. To create the drum information matrix, read the drum track from *data*. If a drum part was hit at that time, store a value of $1$, otherwise store a value of $0$.

It is important to note that each note on the guitar track and bass track is further converted into a 5-dimensional vector. The conversion is as follows ([9])

```
% chroma contains the position of each note in the chroma circle
% 1 indicates the first note in the chroma circle, in this case G
chroma = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
radius_chroma = 1

% c5 contains the position of each note in the circle of fifths
c5 = [1, 8, 3, 10, 5, 12, 7, 2, 9, 4, 11, 6]
radius_c5 = 1
```

```
% 55 is the MIDI value of the note G3
note = mod(midi_note - 55, 12) + 1

% find the angle in the chroma circle and circle of fifths
chroma_angle = (chroma(note) - 1) * (360/12);
c5_angle = (c5(note) - 1) * (360/12);

% compute the (x,y) coordinates for the chroma circle and circle of fifths
% sind stands for sin in degrees
chroma_x = radius_chroma * sind(chroma_angle);
chroma_y = radius_chroma * cosd(chroma_angle);
c5_x = radius_c5 * sind(c5_angle);
c5_y = radius_c5 * cosd(c5_angle);

% n is the distance (in semitones) of midi_note from A4 (69 in MIDI),
% whose frequency os 440 Hz. fx is the frequency of the note
n = midi_note - 69;
fx = 2^(n/12)*440;

% min_note and max_note are the lowest and highest notes
% trained in the network
min_p = 2 * log2(2^((min_note - 69)/12) * 440);
max_p = 2 * log2(2^((max_note - 69)/12) * 440);

% the representation of pitch is scaled in such a way that a pitch
% distance of 1 octave in the first dimension, is equal to the distance of
% notes on the opposite sides on the chroma circle or the circle of fifths
pitch = 2 * log2(fx) - max_p + (max_p - min_p)/2;

% y is the 5-dimensional representation of midi_note
y = [pitch, chroma_x, chroma_y, c5_x, c5_y];
```

However, the output at each step during testing, is a vector of probabilities of selecting each note (and duration) as the next one on the accompanying track. In ESNs an output consisting of discrete values is required.

The input of the network consists of $14$ units: $1$ is a bias input, $1$ contains the rhythm, $5$ units contain the 5-dimensional representation of a guitar note, $1$ unit contains the duration of the respective guitar note, $5$ units contain the 5-dimensional representation of a bass note, and the last $1$ contains the duration of the respective bass note.

Since the output must be discrete, it is represented differently than the input. The output is a vector of size $1 + note\_range + duration$, where $1$ is the *no-note-played* switch, $note\_range$ is the difference between the highest note and the lowest note in the dataset (note values are based on the MIDI scale - that is a note is an integer between $0 - 127$), and $duration$ has a value of 16 as explained above.

One example is the extreme case where all possible notes are present in the large dataset. This means that the range is $128$ since the highest note has a MIDI value of $127$, and the lowest note has a MIDI value of $0$. Thus the output is a vector of size $1 + 128 + 16 = 145$. Note $G3$ with a MIDI value of $55$ and a duration of $\frac{1}{4} = \frac{4}{16}$ is represented by an output vector $[0 \dots 0\,1\,0 \dots 0\,1\,0 \dots 0]$, with the first $1$ being at the $55+1 = 60$ position (indicating the note) and the second $1$ being at the $129 + 4 = 133$ position (indicating the duration).

Another example is the absence of a note, or simply put, the time where an instrument does **not** play. This is represented by the output vector $[1\ 0\ \dots 0]$. Note that the duration of a 'no-note-played' event is not required, since the first output dimension contains the *no-note-played* switch. The MIDI format writes *only* notes that are present in a song. A 'no-note-played' event in the MIDI format is simply not written (or needed). MIDI calculates time from the MTC (MIDI Time Code), an absolute time clock that starts from the beginning of the song or reference point. Therefore, the duration of a note is not an explicit information in the MIDI format. This information was converted to an explicit number only for the purpose of this thesis. Additionally, the output vector $[1\ 0\ \dots 0]$ is simply not written onto the MIDI version of the network bass output, because it is not needed.

After the tracks have been converted into their respective matrix, the data is ready to be driven into the neural network.

## 4.2 Network Parameters

In order to determine the best parameters, further research and trials must be undertaken. This is a complex task, that requires proper dedication. The reservoir acts as **(i)** as a non-linear expansion and **(ii)** as a memory of input $\mathbf{u}(n)$ at the same time [8]. Relying heavily on [8], this practical approach was taken

1. Size of the reservoir. The bigger, the better its performance, provided appropriate regularization measures are taken against overfitting. The bigger the space of reservoir signals $\mathbf{x}(n)$, the easier it is to find a linear combination of the signals to approximate $\mathbf{y}_{teacher}(n)$. Practically, this falls under the computational capacity of your machine. In this case sizes of $100, 200, 500,$ and $1000$ neurons were tested. The network started with $100$ neurons and only proceeded to a higher number when its global parameters were tested. After optimizing a smaller reservoir, scale to a bigger one.

2. The amount of training data used depends on the computational power of the machine. In this case, the data used was always split $70 - 30$ for training-testing. Portions of the data used for training started from $\frac{1}{20}$ of the entire dataset, and were gradually increased to the entire dataset. Since the original data contains more than $7$ hours of music, it is time-consuming to train all of it while tuning the global parameters.

3. Generate a dense $\mathbf{W}^{in}$. Input scaling determines how non-linear the reservoir responses are. For very linear tasks, $\mathbf{W}^{in}$ should be small, letting units operate around $0$ where the activation $tanh$ is virtually linear. If different input channels contribute differently, it is advised to scale them separately. Such is the case here, where the input $\mathbf{u}(n)$ contains data regarding the lead track, bass track, and drum track. $\mathbf{W}^{in}$ is generated with uniformly distributed random numbers from the interval $[-1\ 1]$, whereas $\mathbf{W}$ and $\mathbf{W}^{fb}$ are generated with uniformly distributed random numbers from the interval $[0\ 1]$. The scaling of $\mathbf{W}^{in}$ determines the proportion of how much the current state $\mathbf{x}(n)$ depends on the current input $\mathbf{u}(n)$.

4. Normalize the input. All input channels were normalized and shifted to $[-1\ 1]$.
5. Generate a sparse $\mathbf{W}$. The scaling of $\mathbf{W}$ determines the proportion of how much the current state $\mathbf{x}(n)$ depends on the previous state $\mathbf{x}(n-1)$. An initial reference point is a spectral radius of $1$. The spectral radius determines how fast the influence of an input dies out in a reservoir with time, and how stable the reservoir activations are.
6. The leaking rate $\alpha$ of the reservoir nodes in (2) can be regarded as the speed of the reservoir update dynamics discretized in time. A small $\alpha$ may indicate slow dynamics of $\mathbf{x}(n)$, thus increasing the duration of short-term memory in an ESN.

Knowing that randomly generated reservoirs even with the same parameters vary in performance and that good average performance is not found in a very narrow parameter range, a manual approach trying different ranges of one parameter at a time was taken.

## 4.3 Training the network

During training, at every step $n$, as input is fed the $n^{th}$ note on the accompanying track, the $n^{th}$ note on the leading track, and the $n^{th}$ rhythm value. The $(n+1)^{th}$ note of the accompanying track is forced at every time step $n$. By doing so, the network is taught to predict the next note on the accompanying track. The output of duration is coded in the same manner with $16$ neurons. This form of representation is chosen to ensure network output interpretation as a probability vector. I.e, at every time step during testing, each neuron should have a value proportional to the probability of selecting the respective note at that time.

During testing, at every step $n$, as input is fed the $n^{th}$ node of the leading track and the $n^{th}$ beat value. The network produces as an output a hypothesis vector where a note is selected for the accompanying track. The selected note is fed back to the network as the accompanying track input at step $(n+1)$, together with $(n+1)^{th}$ note on the leading track, and the $(n+1)^{th}$ beat value. The process is repeated until the end of the testing sequence.

## 5 Conducted Experiments

Since ESNs include elements generated randomly, it is important that every random-number-generator function is seeded with the same *seed*. This ensures that *only* changes to global parameters result in changes of performance, and *not* the changes of randomly-generated matrices. Furthermore, dealing with high dimensional data results in poor human intuition. Therefore, it is advised to plot different states of the network, at different times. Following $4.2$, one parameter will change at a time. All experimental results start with $100$ neurons, unless mentioned otherwise. All experiments favored the deterministic choice for selecting the next output, that is, the highest value of the output probability vector was selected.

The *raw* training input was first normalized and then stored into a vector $\mathbf{u}(n)$ for $n$ timesteps. This ensured that the network was trained within bounds of $[-1\ 1]$. Other bounds can be chosen, but they must be consistent within the ESN. If the network is trained on raw data, this can lead to malfunctions. The following graphs show normalized and raw data of the input bass track, the input drum track, and the input guitar track, during the same $timesteps$.

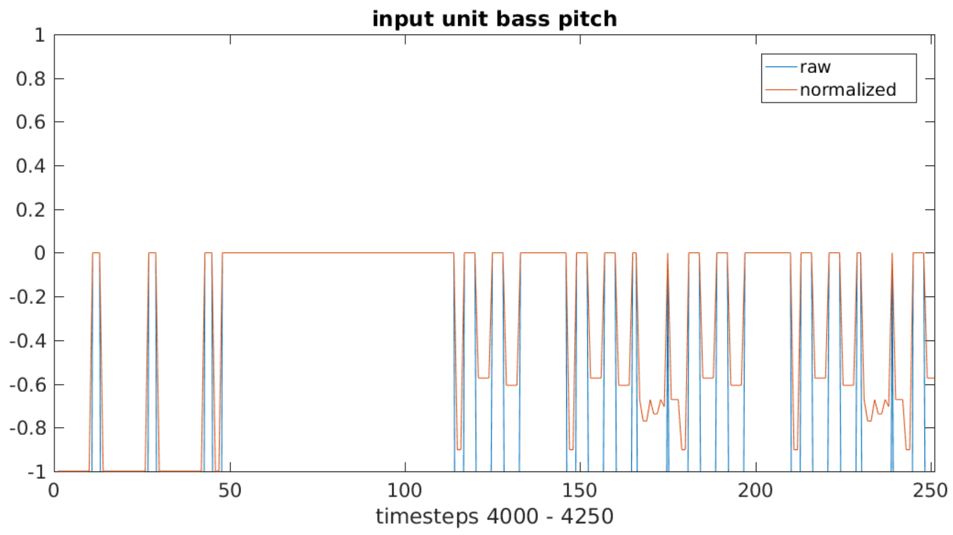

Figure 4: Bass input - normalized vs. raw

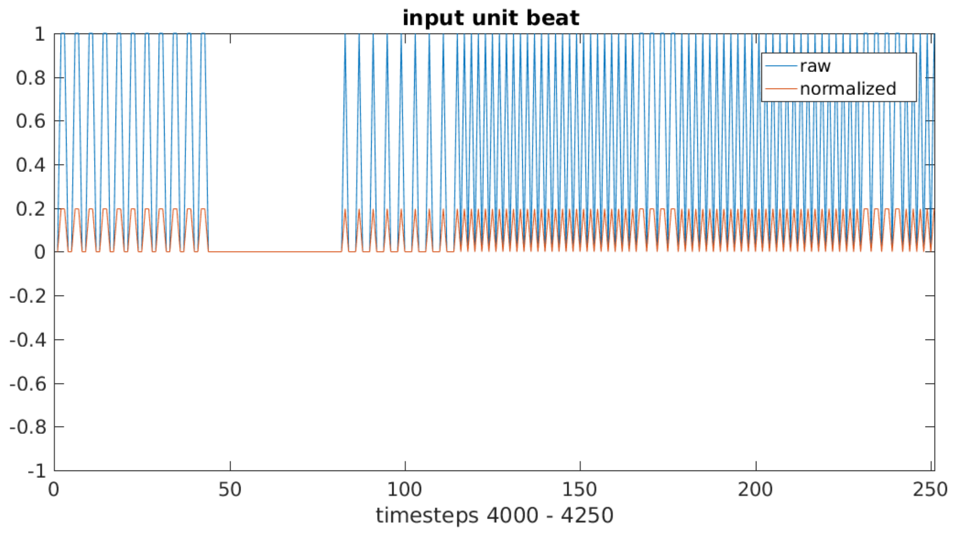

Figure 5: Drums input - normalized vs. raw

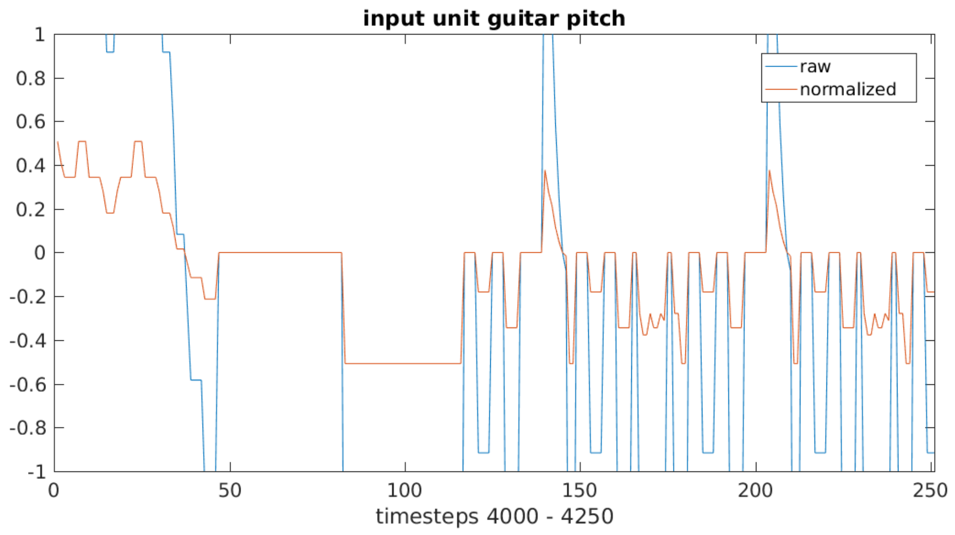

Figure 6: Guitar input - normalized vs. raw

As mentioned in Section $4.2$, $\mathbf{W}^{in}$ should be generated densely. In addition, the drums and bass input channels were scaled by a factor of $0.8$, allowing the network to rely more on the guitar track since, it is assumed, the guitar track has a higher influence on the bass track.

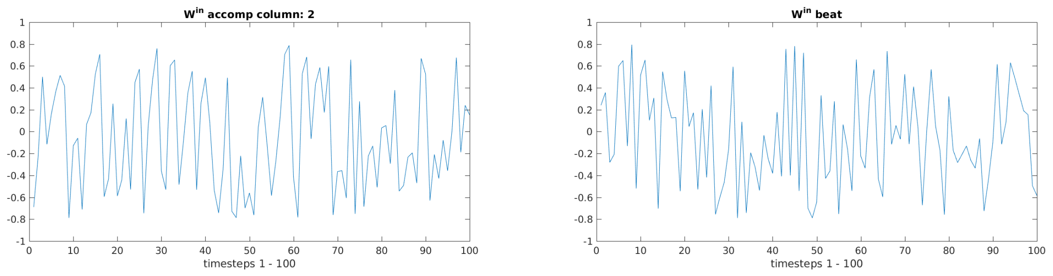

Figure 7: $\mathbf{W}^{in}$ activations

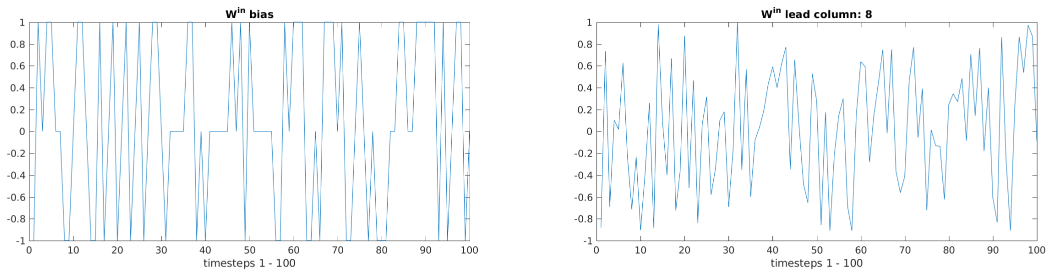

Figure 8: $\mathbf{W}^{in}$ activations

However, $\mathbf{W}$ should be generated sparsely, typically around $0$. It is important to note that the scaling in the plots below has a range of $[-0.2\ 0.2]$. It is also important to note that $\mathbf{W}$ has a uniform distribution $[0\ 1]$, unlike $\mathbf{W}^{in}$ which has a uniform distribution $[-1\ 1]$.

Generating $\mathbf{W}$ between $[-1\ 1]$ led to instabilities.

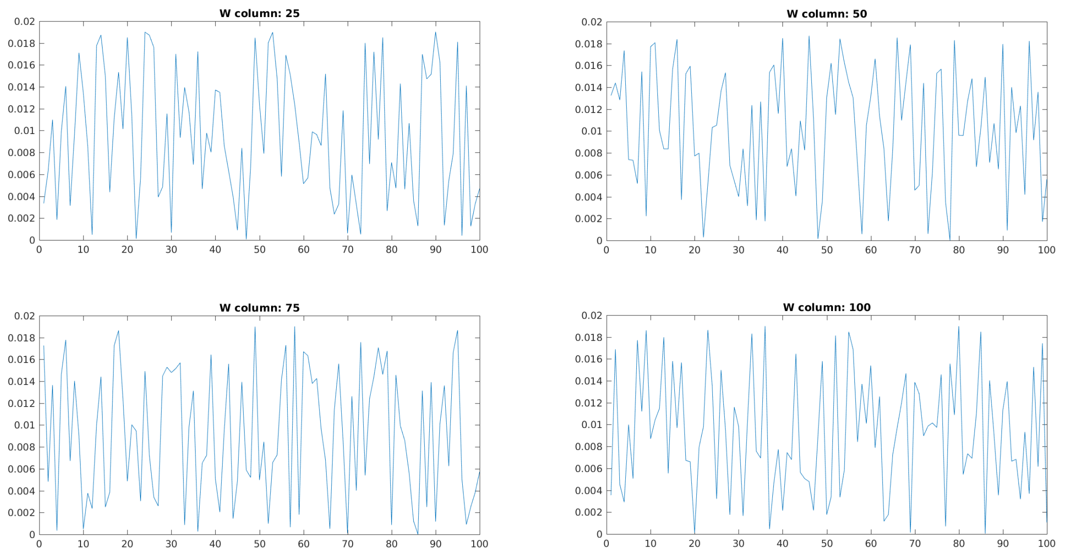

Figure 9: $\mathbf{W}$ activations

Arguably the most crucial parts of an ESN are its activations $\mathbf{x}(n)$. Continuous plotting of the activation states of different activation units (neurons), will aid the network diagnose during experimentation. The aim is to obtain activations that visit different areas of the plot, slightly resembling periodic functions with high frequency.

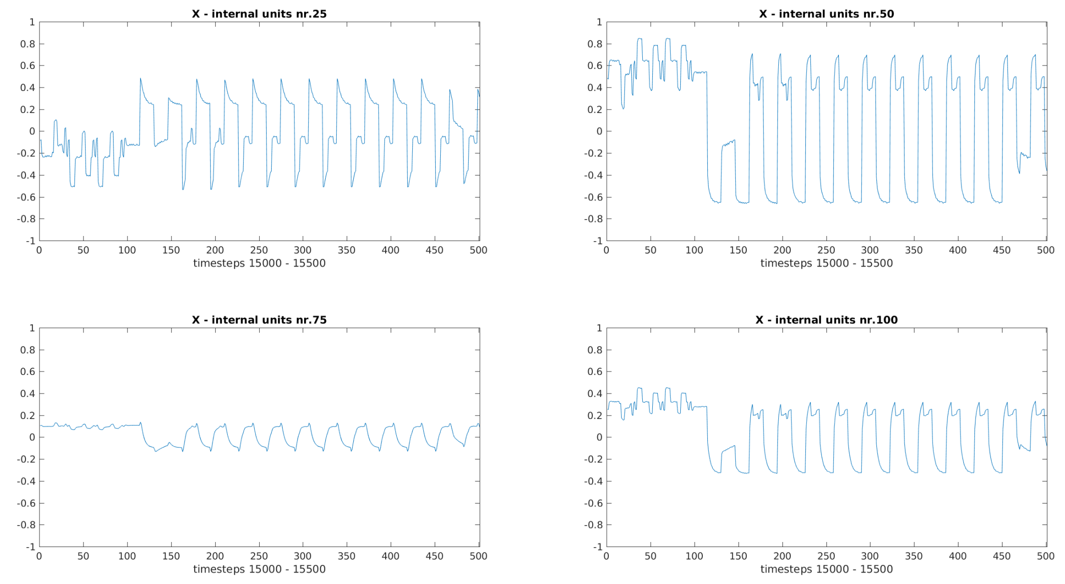

Figure 10: $\mathbf{X}$ activations

The following parameters have yielded best results

1. $spectral\_radius = 0.95$
2. $\alpha = 0.9$
3. $\mathbf{W}^{in}\_scale = 1$

This setup already produced great results. The data is divided in $timesteps$. This is an arbitrary choice, thus for simplicity let us just give $timesteps$ its value from the data set. All data contains $timesteps = 210000$. At first, $timesteps = 40000$ were used. The training-testing ration was $70\% - 30\%$ during all experiments. Data was collected and merged by artist, thus $AC/DC$ was present in the first $75000$ timesteps. This means that the network just trained with $AC/DC$ songs, was tested on new $AC/DC$ songs. The output produced a very pleasing and artistic bass output, while maintaining the $AC/DC$ characteristic. This was believed to happen due to the simple and periodic bass melody of all $AC/DC$ songs. Furthermore, the output kept the rhythm, and smoothly changed it while songs switched. One explanation could be the two empty bars between two songs in the dataset, which were purposely inserted to train a smooth transition and for the network to understand when a particular song is over.

All new reservoir activations $\mathbf{x}_{new}(n)$ during testing were stored in a matrix $\mathbf{X}_{new}$. As we can observe, the reservoir neurons during testing provide a rich variety of activations. This is desired in ESNs.

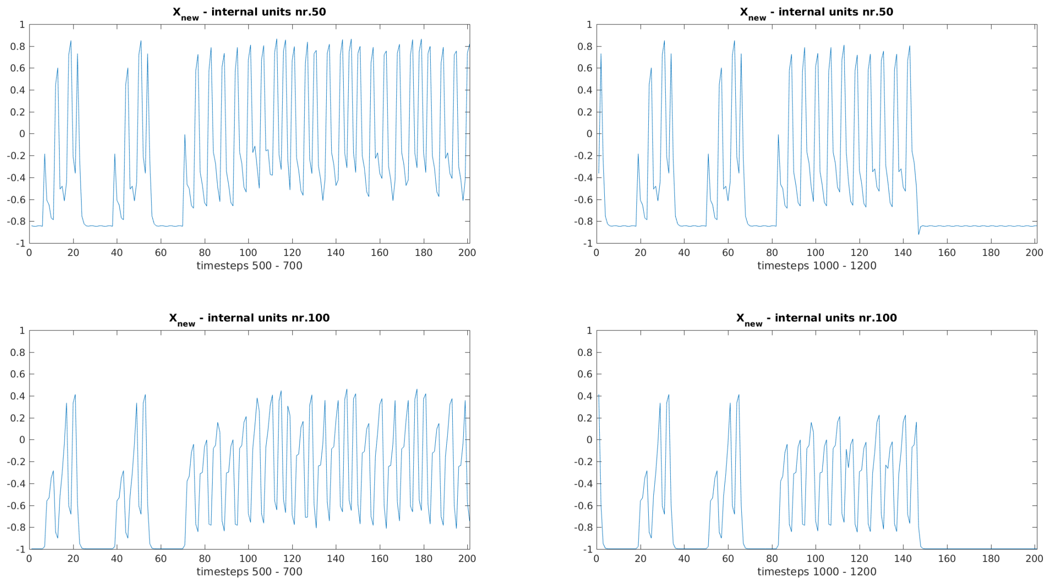

Figure 11: $\mathbf{X}_{new}$ activations

To better help understand the network output, the original bass track from testing is plotted against the network output-bass from testing. Here it should be mentioned that 'network output-bass from testing' means the 'network output-bass' *after* the decision function was applied, i.e not a noisy hypothesis vector, but the output vector as explained in Section $4.1$.

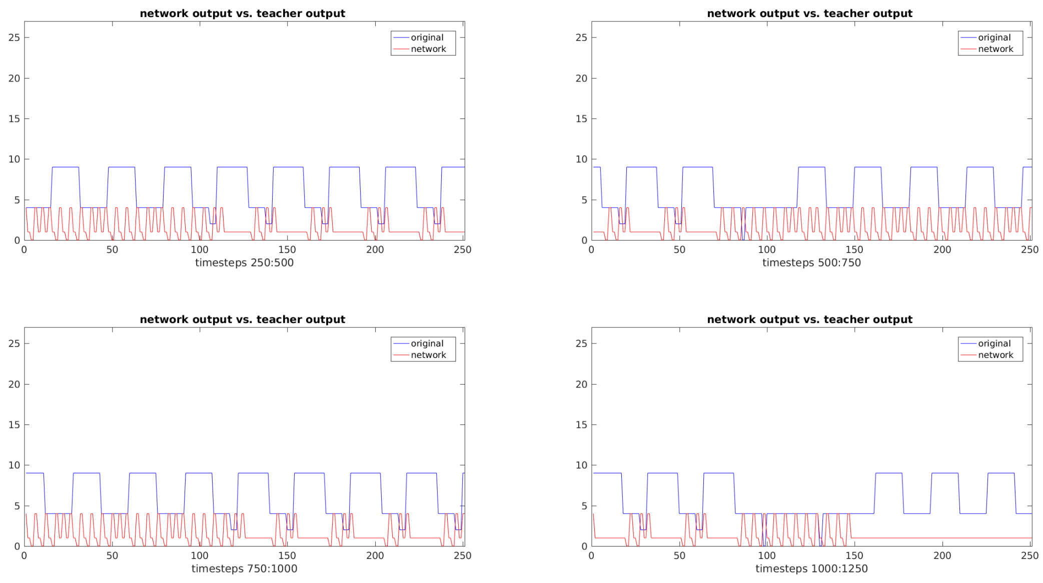

Figure 12: network output vs. real output

However, due to the nature of the network output, not much can be understood from the graphs. As explained in Section $4.2$, the output is a probability vector of size $1 + note\_range + duration$. The $note\_range$ comes from the difference between the highest note and lowest note in the training data. This can lead to testing data that may contain higher (or lower) limits, which in turn leads to a network that is unable to generate a bass melody similar to the original one. In fact, after multiple trials, this has not been an issue. Such is the case in the graphs above, where the testing data does indeed contain higher bass notes. However, the output was still musically pleasing, and it followed the rhythm perfectly. Furthermore, it was observed that $note\_range$ *must* be the range of training data, otherwise the network will misbehave.

Since this network setup already produced great results, more and more $timesteps$ were used, until $timesteps$ reached all training data. This means that $70\%$ of a $7$ hour long dataset was used for training and the remaining $30\%$ was used for testing. The difference between $timesteps = 40000$ and $timesteps = 210000$ was clear. Using the entire dataset resulted in an output that was sometimes absent, sometimes off-key, and sometimes fitting. Two symptoms are believed to have produced this outcome: **(i)** the network was not tuned for such a scale, and **(ii)** the data was arranged by artist. Arranging the training set by artist may yield unexpected results. One example is testing previously-unknown artists. If the dataset is arranged by artist, then the testing partition will contain artists that are unknown to the network. As such, producing a fitting bass melody is more difficult. One solution for this case is cross-validation, but a direct issue is the disproportion of songs per artist. Some artists have more songs in the dataset than others. Because creating the dataset required a significant amount of time, symptom **(i)** was tackled.

After training the network on different numbers of reservoir neurons (network size), it is observed that values above $200$ produce an unstable, off-key output if the dataset is small. On the other hand, values between $100 - 200$ neurons yield best results. However, if the entire dataset is trained with a $70 : 30$ ratio, a network size of $400$ yields a better result, especially after using a feedback connection. Additionally, a 'secret' dataset was used for testing. This dataset includes $10$ songs from different artists present in the large dataset. This 'secret' dataset was *never* used for training. After training the network on the entire $timesteps = 210000$, the 'secret' songs were tested one by one. Here the arrangement problem of the artists was present once again, making it difficult to find a fitting bass melody for the testing songs. A more fitting melody was achieved if the training set included only songs from the same artist (band) as the 'secret' test song.

## 5.1 Supplying two-note chords

So far, the guitar track used for training only contains single notes with their respective duration. This means that whenever a chord (or more than one note at a time) was present, only the lowest note was selected. MIDI representation of a chord starts with the lowest note of that chord. This particular note is usually the root of the chord, especially if the genre of training data is rock. A feature like this proved to be more helpful in training, even though the guitar track was simplified to one note. However, believing that an extension of the guitar track will provide better data for generating an accompanying bass, the network was upgraded to support two-note chords. The current guitar matrix now has 3 rows: the first row includes all single notes plus the lowest note of every chord, the second row includes the second note of a chord (if it exists), the third row includes the duration of the note. Here an assumption is made; notes of a chord have the same duration. A careful reader might ask how the elements of the 'chord' row are filled if no chord was present. Two alternatives are present, **(i)** fill them with zeros (the equivalent of 'no-note'), or **(ii)** fill them with the same note as the the first row. The former yielded better results. Unfortunately, after a lot of trial and error, this extension of the guitar input mislead the network, and produced a worse output.

## 5.2 Including a feedback connection

Music generation by accompaniment requires feedback connections to better train the network. However, dealing with feedback connections is tricky since even the slightest change can drive the network into an unwanted state. After different trials, $\mathbf{W}^{fb}\_scale$ performed better around $0.25$.

# 6 Evaluation Criteria

In order to determine the best parameters, further research must be taken. Tomas divided his evaluation in two parts: one was computing the $log$ of the product of probabilities of selecting the right input at the right time, the other was conducting a survey and run statistical tests on results. While the survey was part of the evaluation, a mathematical evaluation must be further tackled with more literature and more recent approaches. The MSE was not reliable in this task, therefore most of the evaluation was done manually by listening to the output produced by the network.

# 7 Conclusions

The main focus of this guided research is the use of Echo State Networks for a music generation task. Advantages of ESNs make them a perfect fit for this particular task. An ideal outcome with the 'right' evaluation is far from this thesis' reach. However, a musically-pleasing model, trained on a larger set of data, with hopefully better network parameters, has been a desired goal. Due to time constraints, a more detailed experimentation is lacking. This is crucial to better understand the network and its parameters.

## 7.1 Known Drawbacks

1. All data is transposed to the same key.
2. The network only supports three tracks.
3. The drums are represented binary, with a $1$ if the drumkit was hit and $0$ otherwise.
4. The duration of a note is a fraction of $16th$, i.e faster notes are not included.
5. Note information only includes pitch, and duration. Other MIDI information, such as velocity, are disregarded.
6. The dataset contains a disproportionate number of songs per artist.
7. Evaluation criteria relies primarily on manual observation.
8. The $note\_range$ of the testing set is not included in training.

## 7.2 Potential Improvements

First and foremost, a fully comprehensive experimentation is required. This includes different network sizes, tested on different timesteps, with and without feedback connections, generated with different seeds, and tested on single and multiple songs. However, as previously mentioned, in order to understand the impact of a parameter, only one parameter should change at a time. This can exponentially grow ways of conducting experiments, thus a lot of research must be taken before embarking on trial and error. Data should be arranged differently in order to understand its impact on the training dynamics of the network. A better way of reading the drumkit will drastically improve the network. Especially since the bass guitar is usually connected to the rhythm of the bass drum. A better understanding on the impact of certain network parameters is required. Unfortunately, changing a parameter by raising (or lowering) its value will not have the same effect if other parameters are also changed. This means that underneath the deceptively trivial appearance of the network, there exists a more complex correlation between parameters.

### 7.3 Potential Expansions

1. Full chord support
2. Support for more (or all) different keys
3. Support of additional instruments

## 8 Acknowledgements

My sincerest gratitude goes to my supervisor Prof. Dr.Herbert Jaeger for his constant support and guidance, and to my friend Tomas Pllaha for his insight regarding his thesis.

## 9 Appendix

### 9.1 Source Code

Source code with some graphs and sample outputs can be found at github.com/aulon/rapsodi.

## 9.2 List of Data

| | |
|---|---|
| ACDC | Baby Please Don't Go |
| ACDC | Back In Black |
| ACDC | Big Balls |
| ACDC | Can't Stand Still |
| ACDC | Can't Stop Rock N' Roll |
| ACDC | Cover You In Oil |
| ACDC | Dirty Deeds Done Dirt Cheap |
| ACDC | Dog eat dog |
| ACDC | Flick Of The Switch |
| ACDC | For Those About To Rock |
| ACDC | Gimme A Bullet |
| ACDC | Heatseeker |
| ACDC | Hells Bells |
| ACDC | Highway to Hell |
| ACDC | If You Want Blood (You've Got It) |
| ACDC | Inject The Venom |
| ACDC | Jailbreak |
| ACDC | Let There Be Rock |
| ACDC | Moneytalks |
| ACDC | Problem Child |
| ACDC | Rock And Roll Damnation |
| ACDC | Rocker |
| ACDC | Rock 'n' Roll Singer |
| ACDC | Sattelite Blues |
| ACDC | Shoot To Thrill |
| ACDC | Shot Down In Flames |
| ACDC | Sin City |
| ACDC | Sink The Pink |
| ACDC | Stiff Upper Lip |
| ACDC | The Jack |
| ACDC | Thunderstruck |
| ACDC | TNT |
| ACDC | Touch Too Much |
| ACDC | Walk All Over You |
| ACDC | Who Made Who |
| ACDC | You Ain't Got A Hold On Me |
| ACDC | You Shook Me All Night Long |
| Black Sabbath | Behind the Wall of Sleep |
| Black Sabbath | Country Girl |
| Black Sabbath | Electric Funeral |
| Black Sabbath | Hand Of Doom |
| Black Sabbath | Heaven and Hell |
| Black Sabbath | Iron Man |
| Black Sabbath | N.I.B. |
| Black Sabbath | War Pigs |
| Chapman, Tracy | Fast Car |
| Chapman, Tracy | Give Me One Reason |

| | |
|---|---|
| Deep Purple | Burn |
| Deep Purple | Kentucky Woman |
| Deep Purple | Space Truckin’ |
| Def Leppard | Armageddon It |
| Def Leppard | Bringin’ On The Heartbreak |
| Def Leppard | Hysteria |
| Def Leppard | Photograph |
| Def Leppard | Pour Some Sugar On Me |
| Def Leppard | Switch 625 |
| Def Leppard | Women |
| Dio, Ronnie James | Rainbow In The Dark |
| Dio, Ronnie James | Holy Diver |
| Hendrix, Jimi | All Along the Watchower |
| Hendrix, Jimi | Crosstown Traffic |
| Hendrix, Jimi | Fire |
| Hendrix, Jimi | Foxey Lady |
| Hendrix, Jimi | Hey Joe |
| Hendrix, Jimi | Little Wing |
| Hendrix, Jimi | Long Hot Summer Night |
| Hendrix, Jimi | Purple Haze |
| Hendrix, Jimi | Stone Free |
| Hendrix, Jimi | The Wind Cries Mary |
| Hendrix, Jimi | Wait Until Tomorrow |
| Iron Maiden | 2 minutes to midnight |
| Iron Maiden | Hallowed be thy name |
| Iron Maiden | Running Free |
| Iron Maiden | Run To the Hills |
| Iron Maiden | Sign Of the Cross |
| Iron Maiden | The Evil That Men Do |
| Iron Maiden | The Number Of The Beast (guitar pro) |
| Iron Maiden | Wasted Years |
| Judas Priest | Dreamer Deceiver |
| Judas Priest | Hell Bent For Leather |
| Judas Priest | Reckless |
| Judas Priest | Rock Hard Ride Free |
| Judas Priest | The Hellion |
| Judas Priest | You’ve Got Another Thing Comin’ |
| Lynyrd Skynyrd | Free Bird |
| Lynyrd Skynyrd | Gimme Three Steps |
| Lynyrd Skynyrd | Simple Man |
| Lynyrd Skynyrd | Sweet Home Alabama |
| Lynyrd Skynyrd | Tuesday’s Gone |

| | |
|---|---|
| ZZ Top | Beer Drinkers Hell Raisers |
| ZZ Top | Cheap Sunglasses |
| ZZ Top | Got Me Under Pressure |
| ZZ Top | I'm Bad, I'm Nationwide |
| ZZ Top | I Thank You |
| ZZ Top | Just got paid today |
| ZZ Top | Nasty Dogs Funky Kings |
| ZZ Top | Old Man |
| ZZ Top | Planet Of Women |
| ZZ Top | Sharp Dressed Man |
| ZZ Top | Tush |

'Secret' testing data

| | |
|---|---|
| ACDC | It's A Long Way To The Top |
| ACDC | Little Lover |
| ACDC | Whole Lotta Rosie |
| Black Sabbath | Paranoid |
| Deep Purple | Smoke Of The Water |
| Hendrix, Jimi | Voodoo Child |
| Iron Maiden | The Trooper |
| Judas Priest | Breaking the Law |
| ZZ Top | Gimme All Your Lovin' |
| ZZ Top | La Grange |

## References

[1] Shlomo Dubnov, Gerard Assayag, Olivier Lartillot, and Gill Bejerano. Using machine-learning methods for musical style modeling. *Computer*, 36(10):73–80, 2003.

[2] Nicolas Scaringella, Giorgio Zoia, and Daniel Mlynek. Automatic genre classification of music content: a survey. *IEEE Signal Processing Magazine*, 23(2):133–141, 2006.

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

[4] Äaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR abs/1609.03499*, 2016.

[5] Herbert Jaeger and Harald Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *science*, 304(5667):78–80, 2004.

[6] Mantas LukošEvičIus and Herbert Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, 2009.

[7] Herbert Jaeger. The echo state approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148:34, 2001.

[8] Mantas Lukoševičius. A practical guide to applying echo state networks. In *Neural networks: Tricks of the trade*, pages 659–686. Springer, 2012.

[9] Tomas Pllaha. Echo state networks: Music accompaniment by prediction. pages 1–24, 2014.

[10] Roger N Shepard et al. Toward a universal law of generalization for psychological science. *Science*, 237(4820):1317–1323, 1987.

[11] Roger N Shepard. Geometrical approximations to the structure of musical pitch. *Psychological review*, 89(4):305, 1982.

[12] Mantas Lukoševičius, Herbert Jaeger, and Benjamin Schrauwen. Reservoir computing trends. *KI-Künstliche Intelligenz*, 26(4):365–371, 2012.

[13] Herbert Jaeger. *Short term memory in echo state networks*. GMD-Forschungszentrum Informationstechnik, 2001.

[14] Michael A Casey, Remco Veltkamp, Masataka Goto, Marc Leman, Christophe Rhodes, and Malcolm Slaney. Content-based music information retrieval: Current directions and future challenges. *Proceedings of the IEEE*, 96(4):668–696, 2008.

[15] Frederick P Brooks, AL Hopkins, Peter G Neumann, and WV Wright. An experiment in musical composition. *IRE Transactions on Electronic Computers*, (3):175–182, 1957.