# EDAN70 - Project in Artificial Intelligence: Generating Music with Neural Networks

Axel Nyström
Faculty of Engineering, LTH
Lund University
Email: mat04any@student.lu.se

Viktor Stagge
Faculty of Engineering, LTH
Lund University
Email: ine12vst@student.lu.se

***Abstract*—In this project, a Recurrent Neural Network (RNN) was used to generate classical piano music in MIDI format. This works by a two-step process, where a preprocessing stage first transforms the music into a suitable format on which the RNN can subsequently operate. In the preprocessing stage, the music is mapped onto a text format in such a way that all the MIDI files in the training data is represented as a sequence of characters, and any sequence of characters can be translated back into a proper MIDI file. In the second stage, an RNN is trained to predict the next character of such a text, given a sequence of preceding characters. By recursively feeding this prediction back to the RNN, novel music can be created character-by-character.**

**It was found that the quality of the RNN and it's ability to learn interesting aspects of the music was quite sensitive to the way in which the music was preprocessed into text. Several different attempts were made before the network could learn anything that even resembled music. However, once such a mapping was found, structures such as tempo, chords and melodies were learned by the network. Furthermore, it was found that embedding information about the composer into the training data allowed the RNN to distinguish between and mimic different styles of play. Ultimately, while the output produced was decent for short durations, the RNN seems to be unable to learn long-term temporal patterns.**

## I. Introduction

Music, although older than written language (flutes made from animal bones have been found in Europe dating back at least 35000 years), appears to elude precise definition [1]. Yet one does not have to listen to white noise for any longer durations to conclude that, lack of definition notwithstanding, not all sounds qualify as music. Indeed, even listening to a beginner practicing an instrument should be sufficient to infer that some representations of music are "better" than others, regardless of whether the measure by which we come to this conclusion is perfectly understood.

We observe that whatever the definition might be, music must at least be some arrangement of sound in time, following some kind of pattern. This viewpoint, although general, practically begs the question of whether such patterns can be detected and exploited programmatically to analyze or even produce novel music.

In this project, we asked ourselves whether machine learning techniques, being essentially tools for pattern recognition, could be used to generate new music. In particular, we attempt to use a special type of Neural Network called Long Short-Term Memory Recurrent Neural Networks (LSTM-RNN) to train on and create music sequentially, piece-by-piece. For simplicity, we limited ourselves to classical piano music in MIDI-format.

The inspiration for the project was a blog post written by Andrej Karpathy, in which a framework was presented that utilized RNNs to generate text on a character-by-character basis [2]. The networks would train on large amounts of text and essentially learn a set of probability distributions over characters conditioned on a sequence of preceding letters. By sampling from this distribution and feeding the result back to the network, a new sequence can be produced. In this way, the RNN is completely agnostic of concepts such as words, spelling, punctuation and grammar, in the sense that no such information is explicitly encoded into the network. Yet even with a relatively small amount of training, the RNN is capable of producing novel output that is strikingly good at following such rules.

Using this framework, we were able to convert music from our training data into a suitable format with which the RNNs could be trained. The resulting networks were capable of producing music that at least superficially resembled the training data. Certain structures such as tempo, chord progression and melodies were learned quite well for short sequences (up to about 5 seconds), but recurring themes, common time signatures or keys were not. Encoding information about the composer of the various pieces of the training data seemed to enable the network to "imitate" the style of different composers, sometimes with remarkable similarities.

It was found that in general, larger networks training on as much data as possible were able to produce more interesting music. However, training large networks on a lot of data is extremely computer intensive and can easily be prohibitively so. Due to lack of super-computers for this project, the extent to which larger networks scales indefinitely towards more enjoyable music is unknown.

One of the key learning points during the project was the importance of preprocessing the data. It turned out that the results were quite sensitive to the method with which we converted the music files into something that the RNN framework could train on. Certain data representations, although equally expressive in principle, appeared to obfuscate the patterns we were interested in revealing, so much so that several of our first attempts produced nothing that even resembled music.

The remainder of this report is outlined as follows. Section II will cover some of the theory behind Neural Networks and how they relate to Recurrent Neural Networks and Long Short-Term Memory networks. In section III we will explain what the framework by Karpathy does and how it can be used. The majority of our efforts went into preprocessing music so that it could be applied to this RNN framework, and we will go through our different attempts in some detail in section IV. An overview of the various configurations that were tested in our attempts to optimize the performance of the RNN is given in section V, and the results of these tests are covered in section VI. Finally, in section VII, we discuss some potential improvements which, if we had had more time, we would have liked to try. Also in section VII is a brief overview of related work in using Artificial Intelligence to generate music.

## II. Theory

A Recurrent Neural Network is a class of Neural Networks that uses a sequential input, where the output of the network is used as the input for the next iteration [2]. The network therefore learns from its previous prediction, which in turn can be used to more accurately predict the next value. The network is however still limited to how large the input window for the sequence is, which inhibit the RNN from using any contextual information that goes beyond this size. A larger amount of input nodes results in a higher complexity, which in turn increases the time taken to train the network.

To increase the span of how far a RNN can remember a context without increasing its input size, so-called *Long Short-Term Memory cells* can be used. These rely on a hidden state, to which information is added and removed for each input during training [3]. This is done by using three gates referred to as *forget gate*, *input gate* and *output gate*. Each gate has its own weights that are vector multiplied against the previous output and the current input. These are then fitted on the interval $[0, 1]$ using the sigmoid function.

For any input, the forget gate will remove a portion of the saved state. This corresponds to erasing previously learned contexts, that after the given input no longer is active, or active to a lesser degree. The input gate will then combine its own state vector with the previous (weighted) output to determine what information will be added to the state. The expected output is calculated, and then weighted against the LSTM current state. The update equations for the gates are shown in (1).

$$\begin{aligned} f_t &= \sigma(W_f * [h_{t-1}, x_t] + b_f) \\ i_t &= \sigma(W_i * [h_{t-1}, x_t] + b_i) \\ C_t &= \tanh(W_c * [h_{t-1}, x_t] + b_C) * i_t + f_t * C_{t-1} \\ o_t &= \sigma(W_o * [h_{t-1}, x_t] + b_o) * \tanh(C_t) \end{aligned} \tag{1}$$

In order to allow for deeper learning, multiple layers of RNN can be stacked on top of each other, so that the outputs from one layer becomes the inputs for the next layer.

## III. The Framework

The framework used in this report was implemented in LUA by Andrej Karpathy [4]. It allows for training RNNs of different sizes and with various hyper-parameters, by providing training data in the form of text. The text is read character-by-character, where characters are defined by the Unicode standard, and the vocabulary or alphabet used becomes all the characters that were encountered in the training set. Once trained, such a network essentially becomes a machine that is able to estimate the conditional probability mass function $p$ of a character from the alphabet $\mathcal{A}$, given a sequence $\boldsymbol{X}$ of previous characters. That is, given the sequence $\boldsymbol{X} = x_1 x_2 ... x_{n-1}, \quad x_i \in \mathcal{A}$, we obtain a probability mass function $p_{x_n|\boldsymbol{X}}(x)$. By sampling from this distribution we can guess the next character in the sequence and then recursively feed the resulting new sequence into the machine to obtain the next character, producing a new sequence which then gives a new character, et cetera. In this way, the RNN can generate an infinite sequence of text, character-by-character, that in some sense resembles (hopefully) the text on which it was trained.

As an example of what this may look like, listing 1 shows a text produced by a 3-layer RNN using $512$ hidden nodes on each layer, after having trained for several hours on the complete works of William Shakespeare.

Listing 1. *Excerpt from RNN trained on Shakespeare.*

```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.
```

## IV. Applying the Framework to Music

### A. Limiting the Scope

We decided to limit our data set to classical piano music, since it seemed reasonable to expect that music involving a single instrument would be easier to approach than that of a symphony orchestra, say. A lot of good training data is also available in the public domain due to their age. In order to further facilitate our needs, the standard MIDI file format was chosen. MIDI (short for Musical Instrument Digital Interface) is useful for our purposes because it abstracts away much of the underlying complexity that we don't necessarily care about when we think about music. A MIDI file doesn't contain any actual sounds, only information about what notes should be played and in what order. In this way, it is similar to sheet music. To listen to a MIDI file, some form of synthesizer

is required to actually interpret the instructions in the file as sound.

One consequence of these abstractions is that MIDI files are very compact. While an hour of music in the WAV format takes about $600Mb$, an hour of MIDI music is only around $324kb$.

In order to maintain a similar quality of all the files in the training data, all the material used came from the web page *Classical Piano Midi Page* by Bernd Krueger [5]. The complete data set consists of 328 songs written by 25 different classical composers (Beethoven, Mozart, Liszt, Chopin, to name a few). In total this represents 22.4 hours of piano music and $7.26Mb$ of data in the MIDI format.

### B. The Need for Preprocessing the Music

Before we can use Karpathy's framework to generate music, it is clear that the music in our training set must first be transformed into a text-like format. The alphabet $\mathcal{A}$ is restricted to the set of all bytes, and the music, both input and output, should be in the set $\mathcal{M}$ of all possible MIDI-files. Let $\mathcal{D} := \{(x_n)_{n\in\mathbb{N}} | x_n \in \mathcal{A}\}$ be the set of all byte-sequences. Then clearly $\mathcal{M} \subset \mathcal{D}$. This represents a problem for our RNN, as it's output may not be a valid MIDI-file. It is reasonable to expect that it would be very difficult for an RNN to learn the necessary structures such that it only produces sequences belonging to $\mathcal{M}$. Any "mistake" made by the RNN while producing an output sequence could mean that the output is not valid which then means that it can't be listened to.

What we need, then, is to map the training data, which is a subset of $\mathcal{M}$, onto $\mathcal{D}$ in such a way that the reverse operation can also be applied without losing anything of importance. We want any output from the RNN to be "forced" into becoming a valid MIDI-file, while keeping it from having to learn structures such as header information and so on that are part of the MIDI format. Formally, we want to define two functions, $f : \mathcal{M} \mapsto \mathcal{D}$ and $g : \mathcal{D} \mapsto \mathcal{M}$, such that

$$\begin{aligned} (f \circ g)(d) &= d, \quad \forall d \in \mathcal{D} \\ (g \circ f)(m) &= m, \quad \forall m \in \mathcal{M}_{piano} \end{aligned} \tag{2}$$

where $\mathcal{M}_{piano} \subset \mathcal{M}$ is the set of one-track MIDI-files of piano music, to which our training data belongs.

### C. The MIDI and CSV Formats

Of course the preprocessing functions $f$ and $g$ can be defined in any number of ways, and as it turns out, the details of how this is done plays a critical role in the efficiency with which the RNN can find useful patterns in the music. The first step in processing our training data was to convert the MIDI-files into Comma Separated Values (CSV), which is much more readable for humans. This was done using a freeware program called *MIDICSV* [6]. Although easier to understand, the CSV-format contains a lot of redundant characters, and is also following a very specific syntax. Not all text files are valid CSV files.

The MIDI (and CSV) format is essentially just a long list of instructions to the computer. With the exception of some header information, tempo instructions and various sound effects like echo, vibrato, panning, etc., each row in the CSV file tells the computer to either turn a note on or off at some particular time, along with the pitch and volume and track of that note. Starting and stopping a note is encoded in two separate instructions. The instructions are sorted in ascending order by when they should be executed. The timing is counted in cycles, where the duration of a cycle is defined by two instructions in the header of the file. The first header instruction specifies the number of cycles per quarter note, and the tempo instruction, which can be changed later in the file, specifies the number of microseconds per quarter note.

As can be seen in the CSV example in listing 2, which is a short snippet from one of the songs in our training data, the tempo is changed very often. This is a result of the training data being recorded "live": the MIDI files that we used were constructed by a human playing the different songs on an electric keyboard recording the instructions. In order to obtain maximum precision in the timing of each note, the tempo is changed in complicated ways. This seemed to us like it was adding unnecessary complexity to the data format, so we decided to first smooth all the data by removing spurious tempo changes and recalculating the time instructions accordingly. Because of rounding errors, this may have adjusted some note durations and timings by up to approximately half a millisecond, which seemed to be a fair trade-off.

Listing 2. *Excerpt from CSV file.*

```
1, 10120, Tempo, 345821
1, 10160, Tempo, 344828
1, 10160, Note_on_c, 0, 74, 0
1, 10160, Note_on_c, 0, 76, 28
1, 10180, Tempo, 343840
1, 10180, Control_c, 0, 64, 127
1, 10220, Tempo, 342857
1, 10240, Note_on_c, 0, 76, 0
1, 10240, Note_on_c, 0, 74, 32
1, 10260, Tempo, 341880
1, 10280, Tempo, 340909
1, 10320, Tempo, 339943
1, 10320, Note_on_c, 0, 74, 0
1, 10320, Note_on_c, 0, 76, 28
1, 10360, Tempo, 338983
1, 10380, Tempo, 338028
1, 10400, Note_on_c, 0, 76, 0
1, 10400, Note_on_c, 0, 74, 32
```

### D. Three Attempts

The basic approach we used for converting the CSV files into "text" was to let several characters form "words", where each word corresponds to a MIDI/CSV instruction. A CSV instruction would then be decomposed into smaller parts, each of which could be represented by a single character in our alphabet $\mathcal{A}$, and depending on the position of the character within the word, that character would convey different meaning. Such a sequence of characters could then be converted back into a CSV and ultimately MIDI format by adding appropriate header information and then translating the characters into corresponding instructions based on their position within each word.

The first attempt using this format would consider only three things in each CSV instruction: the relative time since the last instruction (represented by two characters), whether the note is turned on or off (one character) and the pitch of the note (one character). This corresponds to the format d0.1 in table I.

The MIDI data corresponding to volume was omitted and a medium value of 68 was assumed. Thus one instruction in the CSV file corresponded to a word made up of 4 characters in our data representation.

The output of a network trained on data like this sounded rather similar to a person falling asleep on the piano. A number of notes were played, and then never stopped playing. It turned out that the RNN had serious problems with learning that once a note is turned on, it might be a good idea to turn it off at some point in the near future.

Learning from our mistakes, in the second format, called d0.2 in table I, we combined all pairs of instructions in the CSV format that were responsible for turning a note on and off. Now our format encoded the relative time since the last instruction (same as before), the duration of the note (two characters) and finally the pitch of the note (same as before). So now each "word" consisted of five characters.

The results from a network trained on this format were unfortunately not much better than before. Although notes were no longer played indefinitely as before, most of the durations involved still appeared to be extremely long. It seemed as though the output of the RNN was getting out of sync. The network could basically be playing perfect Mozart, but if all the output were offset by one byte, it would sound completely wrong. There was simply no way of determining where one word begins and another ends.

In order to correct for this, we inserted a unique separator character in between every five characters of the training data. This represents format d1 in table I. As we had hoped, it didn't take long for the RNN to learn this very regular pattern (compare to how the RNN in the Shakespeare example would learn to regularly place spaces between words). When we now split the output of the network on where the separator characters were, we could simply discard words that weren't exactly the right length. This turned out to make all the difference in the world: the output of the network was now miles above what it had been previously, and in many cases quite listenable. In a sense it seemed comparable to the Shakespeare example but for music: superficially impressive, sometimes really good for very short durations, but ultimately nonsensical.

## V. Optimizing the Performance

Training an RNN is very computer intensive. With our limited computational resources, a training session would take anywhere between 5 and 35 hours before the loss function began to indicate diminishing returns, and in most cases further training might have been possible by adjusting hyper-parameters such as learning rate and dropout. In any case, although there were many questions in regards to how good results we would be able to obtain with this framework, we were fundamentally limited by the number of tests we had time to execute. In order to try to maximize the amount of information we obtained from each training session, we divided our inquiries into four different categories:

| Format | Data representation: byte\|byte\|byte\|byte\|byte |
|---|---|
| *d0.1* | dt ǀ dt ǀ on/off ǀ tone |
| *d0.2* | dt ǀ dt ǀ tone ǀ duration ǀ duration |
| *d1* | dt ǀ dt ǀ tone ǀ dur. ǀ dur. ǀ *separator* |
| *d2* | dt ǀ dt ǀ tone ǀ dur. ǀ dur. ǀ velocity ǀ *separator* |
| *d3* | dt ǀ dt ǀ tone ǀ dur. ǀ dur. ǀ composer ǀ *separator* |
| *d4* | dt ǀ dt ǀ tone ǀ dur. ǀ dur. ǀ v. ǀ c. ǀ *separator* |

TABLE I

*Different data formats that were used to train the networks. Each label represents one byte, two bytes were used where a label occurs twice. The d1 format was the first "successful" one, and d4 is the most expressive, encoding both velocity and composer information in each MIDI instruction.*

| Data sets | |
|---|---|
| *Mixed* | 50 songs, randomly selected from entire data set |
| *Medium* | 71 songs, only Beethoven, Mozart and Haydn |
| *Large* | Entire data set, 328 songs |
| *Mozart* | Only Mozart, 21 songs |
| *Beethoven* | Only Beethoven, 29 songs |

TABLE II

*Different subsets of our training data, used for various testing purposes.*

1) Network size - How many layers should we use, how long should the sequence length be and what is a good size for the hidden layers?
2) Training parameters - What are good learning rates, decay rates, drop out rates and so on?
3) Data sets - How does the choice of training data affect the outcome? In particular, is it better to use music from different composers, or should we train only on Mozart, for instance?
4) Data representations - We can encode more information into our data representation, in particular velocity and composer information. How does this affect the outcome?

While these questions most likely depend to some degree on each other, we decided to treat them independently. This was done by creating a number of canonical data representations, data sets and network sizes, which are summarized in tables I, II and III respectively. Using these categories, a number of experiments were specified as can be seen in table IV. Since it is not clear what exactly is meant by one network being better than another at composing music, we resorted to using our own subjective listening pleasure when judging the quality of the output of the different networks. Although not entirely scientific, in this way we were at least able to get some indication as to which approaches were more promising.

## VI. Results

Comparisons between the outputs of sessions 2, 3, 4 and 5 seemed to suggest that the medium data set was slightly better, presumably because it contained more data to train on. It was

| Network sizes | Layers | Sequence length | Hidden layer size |
|---|---|---|---|
| *Small* | 2 | 150 | 128 |
| *Medium* | 2 | 150 | 256 |
| *Large* | 3 | 300 | 350 |

TABLE III

*Different network sizes that were tested.*

| Test ID | Network size | Data set | Data format |
|---|---|---|---|
| 1 | medium | medium | d1 |
| 2 | small | medium | d1 |
| 3 | small | beethoven | d1 |
| 4 | small | mozart | d1 |
| 5 | small | mixed | d1 |
| 7 | small | medium | d2 |
| 8 | small | medium | d3 |
| 9 | small | medium | d4 |
| 12 | large | large | d4 |

TABLE IV
*The different training sessions that were tried.*

interesting to note that the styles of different composers would shine through to some extent: the network trained exclusively on Beethoven seemed a fair bit more radical and explosive, while the Mozart network appeared a bit more structured, reflecting the general temperament of the pieces in the training data. The network trained on mixed data produced a quite disparate output, probably as a result of the diverse styles of the different composers. From these reflexions we concluded that more training data is probably always better, and that perhaps the radical switching between different styles could be prevented by encoding composer information into the data representation, as was done in formats d3 and d4.

Comparing the outputs of sessions 2, 7, 8 and 9, which differed only in the data representation, it was pretty clear that the more expressive d4 format performed better. Since each instruction was $40\%$ longer in this format compared to d1, one might have expected that the network would require longer learning times, or produce less enjoyable music, but as far as we could tell, this was not the case. Instead, encoding the velocity seemed to allow for more dynamical playing without a particularly high cost. The composer information appeared to make the network output less erratic, though at this point it was a difficult to really tell the difference between the three different composers.

The best results were produced by session 12, using the largest network on the entire data set with the most expressive data representation. Compared to session 1, which was our second largest network trained on format d1 with the medium data set, session 12 was able to play for long times, upwards of 10 minutes, mimicking the style of a particular composer, before switching to something else. Although some of these "styles" were quite awful, when compared to the training data of the relevant composers it was often pretty obvious where the inspiration was coming from. The network was essentially producing nonsensical parodies of Albéniz, Liszt, Beethoven, etc.

When it comes to the fine-tuning of hyper-parameters, the only conclusion we could really draw was that it was difficult to draw any conclusions. The common practice of reducing the learning rate as training continued seemed valid, but what the actual decay rate ought to be was more difficult to assess. Similarly, increasing the dropout rate seemed to allow for lower loss rates, but at the cost of significantly longer training times.

## VII. Discussion

### A. Further Improvements

Although we are generally pleased with the outcome of the project, there are of course many improvements that could still be made. In particular, we struggled with finding good hyper-parameters for training the networks, or rather, it was difficult to really evaluate the effect of different parameters due to the long training times. More training data and better computer equipment would also likely improve the results, especially if GPU acceleration would have been utilized.

Various modifications of the LSTM-structure in the equations (1) could also be envisioned, which may or may not have led to further improvements. For example, it is possible to combine the input and forget gates into a single update gate, which might lead to better results. So-called peephole connections may also be added, and it would have been interesting to experiment further with such variations.

It is possible that combining multiple networks trained on different aspects of the music could have been a useful approach, although this was not tested. For example, perhaps one network could be responsible for deciding the timing of the notes, thus representing the temporal aspects of the music, while a different network representing the harmonic aspects of the music would, in conjunction with the first, decide on the actual notes to be played. Although in theory a single network might have the same expressive power, "hard-coding" such structure into the problem might help with finding the desired patterns.

### B. Related Work

The sequential nature of the RNN, especially with our data representation, poses a problem whenever multiple notes are played simultaneously in chords. There is a combinatorial explosion of equivalent formulations, which makes it much more difficult to accurately represent polyphonic music. Boulanger-Lewandowski et al. were able to overcome this problem, at least in part, by combining an RNN with a Restricted Boltzmann Machine (RBM) which works to reduce the dimensionality of the problem [7].

The problem of evaluating the quality of the output of a generative model for music motivates the usage of what is called Generative Adversarial Networks. In such a model, two networks are compete against each other in a zero-sum game. One network produces music (for example) and the other network tries to determine whether a sample comes from the first network or the training data [8].

A different approach was taken by Johanson and Poli, who created an interactive system with which a genetic algorithm would produce tunes whose fitness were then evaluated by a human user. This framework was then combined with a Neural Network that learned to rate the output of the genetic programming system, although it was not as good as human users [9].

Yet another approach to generating music and sound in general is that of Google's WaveNet. Rather than working

with the abstract descriptions of music found in the MIDI format or similar, the WaveNet approach is to look at the sound waves directly. By combining RNNs with dilated causal convolutions, the attention span of the network was greatly improved. Although primarily aimed at text-to-speech, the same architecture was also successfully used to generate novel, realistic sounding music [10].

Another project worth mentioning in this context is Google's Magenta project, which aims to explore different machine learning techniques for generating music and art. In the process, the hope is to also produce an open-source infrastructure based on TensorFlow with which a community of artists, coders and researchers can collaborate to produce state-of-the-art art, as it were [11].

## VIII. Summary

In this project, we have trained Long Short-Term Memory Recurrent Neural Networks on classical piano music, and used the resulting network to generate new music, character-by-character. The framework for the network was created by Andrej Karpathy and was meant to be used for text, so much of the focus for this project was to apply the framework in the context of music. It was found that preprocessing the training data was critical in obtaining output of any artistic value. In particular, when instructions are encoded as words consisting of a sequence of characters belonging to a common alphabet, it was important to use a special character to separate those words, otherwise the output of the network would easily be misinterpreted.

Our tests suggests that larger networks trained on more data generally produces better music. Furthermore, it appears that the model used was not actually very sensitive to the word length, so it was possible to encode more information about each musical instruction than originally anticipated. Encoding information about the composer into the training data allowed the network to differentiate between various styles of music.

While the output of our best networks would perhaps be able to impress the untrained ear of a layman for a couple of seconds, it is clear that this approach to generating music still has a long way to go before it can produce anything that anyone would want to listen to for reasons other than curiosity. With a bit more computer power, what we achieved in this project might be enough to provide interesting ideas that, with the post-processing of a human, could perhaps be turned into viable music. For a fully automated system however, an all together more sophisticated approach is likely to be necessary.

The scripts used to preprocess the MIDI files, as well as the results of our training sessions and some sample output can be found on the git repository https://github.com/Tipulidae/EDAN70.

## Acknowledgments

We would like to thank Andrej Karpathy for his brilliant introduction to Recurrent Neural Networks, as well as his implementation of a simple-to-use fully functional version of LSTM-RNN. Thank you also to our supervisor Jacek Malec for all the help during the project. And finally, our gratitude and apologies posthumously to all the great composers whose works were defiled in the process of this project.

## References

[1] Conard, Nicholas J., Maria Malina, and Susanne C. Münzel. "New flutes document the earliest musical tradition in southwestern Germany." Nature 460.7256 (2009): 737-740.

[2] Andrej Karpathy. "The Unreasonable Effectiveness of Recurrent Neural Networks." http://karpathy.github.io/2015/05/21/rnn-effectiveness/ Accessed: 2017-06-05

[3] Hochreiter, Sepp, and Jürgen Schmidhuber. "Long short-term memory." Neural computation 9.8 (1997): 1735-1780.

[4] Andrej Karpathy. "Multi-layer Recurrent Neural Networks (LSTM, GRU, RNN) for character-level language models in Torch." https://github.com/karpathy/char-rnn Accessed: 2017-06-05

[5] Bernd Krueger. http://www.piano-midi.de/ Accessed: 2017-06-05

[6] Fourmilab. MIDICSV http://www.fourmilab.ch/webtools/midicsv/ Accessed: 2017-06-05

[7] Boulanger-Lewandowski, Nicolas, Yoshua Bengio, and Pascal Vincent. "Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription." arXiv preprint arXiv:1206.6392 (2012).

[8] Goodfellow, Ian, et al. "Generative adversarial nets." Advances in neural information processing systems. 2014.

[9] Johanson, Brad, and Riccardo Poli. GP-music: An interactive genetic programming system for music generation with automated fitness raters. University of Birmingham, Cognitive Science Research Centre, 1998.

[10] Google's wavenet, generating audio directly as a waveform. Trained also on piano music. van den Oord, Aäron, et al. "Wavenet: A generative model for raw audio." CoRR abs/1609.03499 (2016).

[11] Google's Magenta project. https://magenta.tensorflow.org/welcome-to-magenta Accessed: 2017-06-05