# **Composing Melodies from Polyphonic music using Recurrent Neural Networks**

|  |
| --- |
| Arulselvan Madhavan  Northeastern University  Boston, USA.  Madhavan.ar@husky.neu.edu |

**Abstract**

*In this paper, I present the results of my attempts to compose melodies from polyphonic music using Recurrent Neural Network(RNN) architectures like Long Short Term Memory(LSTM), Gated Recurrent Unit(GRU) and vanilla RNN. Polyphonic music compositions, written in MIDI format, with a harmonic chord sequence track and a melody track were downloaded from* [*http://piano-midi.de*](http://piano-midi.de) *and used as the dataset for this project. The RNNs[[1]](#footnote-1) were fed with harmonic chord sequence and trained to predict melody notes that would sound aesthetically pleasing to human ears when played along with the given chord sequence. In my experiments, LSTM architecture was able to produce more aesthetically producing melodies in comparison with GRU and vanilla RNN.*

# Introduction

Computer aided music generation is a research topic that has been actively explored in earlier decades. Algorithmic music generation task has always been aimed at producing music that are aesthetically pleasing to human ears. To achieve this goal, any algorithmic music generation model should learn the music composition behavior of humans. Musical sequences are heavily dependent on temporal relations. For example, a musical pattern and theme appearing early in one part of a musical piece may appear again several time steps later and many more times throughout the musical piece. So, it is obvious that any algorithmic music generation model must learn the temporal dependencies in a music composition.

## Existing Methods

Existing methods of computer aided music generation models depend on carefully engineering musical features and rely on simple generation schemes such as Markov models or graph based energy minimization techniques. The main problem with existing methods is that they fail to make use of the temporal dependencies in musical sequences. In the realm of machine learning, the best architecture to learn temporal dependencies is Recurrent Neural Networks(RNN). Vanilla RNNs are good for learning short temporal dependencies but musical sequences can be very long depending on the sampling rate used to represent the musical piece. Vanilla RNNs suffer from the problem of vanishing gradients[7] as the length of the sequence increases. In 2012, Nicolas Boulanger-Lewandowski, Yoshua Bengio and Pascal Vincent showed that temporal dependencies in long musical sequences can be modeled without using neural networks by using Restricted Boltzmann Machine [2] to model the temporal dependencies in high dimensional sequences generated from polyphonic music. Later, Douglas Eck and Jurgen Schmidhuber brought back neural networks into musical research by using LSTMs[3] to solve the problem of vanishing gradients and applied it successfully in long musical sequences by generating “blues” music using LSTM.

## What’s new?

In this project, I have attempted to model the long temporal dependencies in polyphonic music using LSTM, GRU and vanilla RNN. Douglas Eck and Jurgen Schmidhuber used LSTM[3] on “blues” music. Polyphonic music[5] consists of two or more simultaneous lines of independent tracks(also known as phrases in music literature), as opposed to a musical texture with just one voice, monophony, or a texture with one dominant melodic voice accompanied by chords, which is called homophony. The polyphonic music used in this project consists of two tracks – harmonic chord sequence and melody track. I used the harmonic chord progression as input to the RNN model and tried to predict the melody notes[8] that will sound melodious to human ears when played along with the given chord sequence. The main use case for these models is in polyphonic transcription, the challenge of predicting the underlying notes of a polyphonic audio signal without access to the underlying score.

## Studied Approach

Figure 1 Neural Network Architecture used

The aim of this project is to compose music notes that will sound aesthetically pleasing to human ears when played along with a chord sequence. So, naturally any musical piece with a chord sequence and corresponding melody notes lends itself as a suitable data for this project. A high level overview of the architecture used in this project is shown in Figure 1. 3 different architectures were studied to compare and contrast their performance with each other.

## Data transformation

### MIDI vs FFT

Our neural network model requires the input data in a digital format(matrix format). Musical sequences can be converted to digital format using 2 methods. One way is to use raw audio files in wav format and apply FFT(Fast Fourier Transform) transformation[4] to get the digital representation. Another way is to use the MIDI representation which is the standard format in which music has to be written in order for it be exchanged between digital music instruments. In this project, I chose to use the MIDI representation of music because an FFT transformation of music captures a lot of complex details of the music, like timbre and spectral information of music and is too sophisticated for this project. In this project, I am only interested in the pitch and duration of a music notes which is captured nicely in a MIDI representation.

### Parsing MIDI files

A MIDI representation of musical piece encodes several information like pitch, time step when the note starts, time step when the note ends, number of tracks in the file etc., The information used in this project to construct a piano roll representation of the musical piece are

1. Note’s pitch: The pitch value tells us which piano key was pressed. For example, a pitch value of 48 tells us that “C4” was pressed.
2. Note ON: The time step(tick) in which the piano key was pressed.
3. Note OFF: The tick in which the piano key was released.

The time difference between the “note\_on” and “note\_off” midi events gives us the duration over which a piano key was held. There are 88 piano keys which could generate 88 piano different piano notes over 7 octaves. However, not all the keys are used in a musical composition. The harmonic chord sequence in the dataset uses 12 piano keys and the melody sequence in the dataset uses 24 piano keys. So, the number of input features(columns) in the input matrix is 12 + 1 for the rest [9] periods and the number of output classes is 24 + 1 for rest periods.

### MIDI to piano-roll Representation

A piano-roll representation is a method of representing a musical stimulus or score for later computational analyses. It can be conceived as a two-dimensional matrix, graph or display. In my piano-roll representation, I used the columns to represent the notes and the rows are the continuous representation of the time steps. A transposed version of piano-roll matrix is shown in Figure 2 for easier understanding.

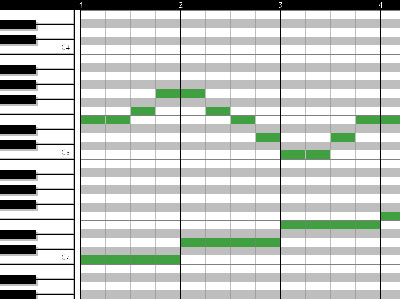


Figure 2 A transposed version of piano roll representation for easier understanding

### Harmonic Chord Sequence

The polyphonic music used in this project has both chord sequence and melody tracks. Harmonic chord sequence can have multiple notes active at any given time step. Figure 3 and Figure 4 are shown as an example of a typical chord sequence where at least 3 different notes are active at any point in time. So, in the piano-roll representation of chord sequence, at any time step(row), multiple notes(columns) will be active (set to 1).



Figure 3 A chord sequence where multiple notes are active at a time.

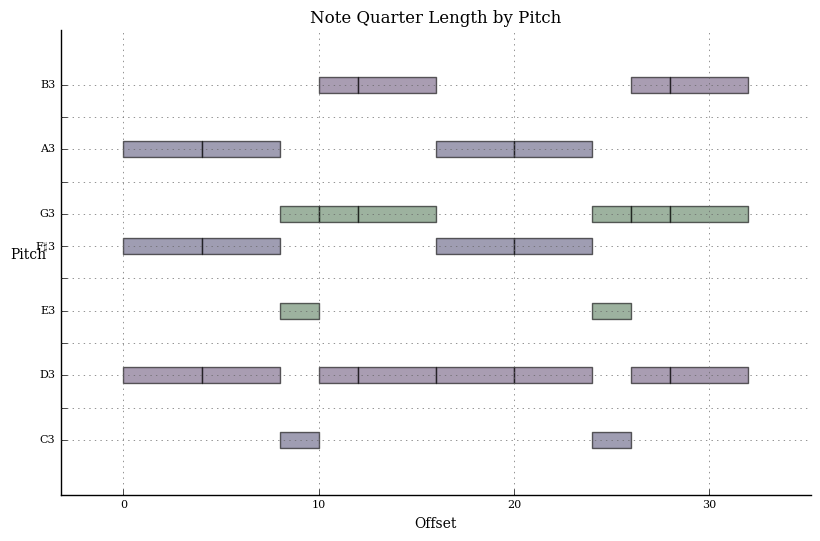


Figure 4 Chord Sequence expressed in Pitch vs time offset

### Melody Sequence

Melody sequences are composed of notes and rests[9]. A time step where no note is played is known as rest. Figure 5 and 6 shows an example melody sequence. Also notice that there is exactly one note played at any time step. So, in the piano-roll representation of melody sequence, a time step(row) can have at most 1 active note. Rest periods will have all zeroes. Since Rest period is not a valid piano-note, an additional column has to be added to the piano-roll matrix to represent the Rest periods in a musical piece.



Figure 5 Melody sequence showing notes and rest

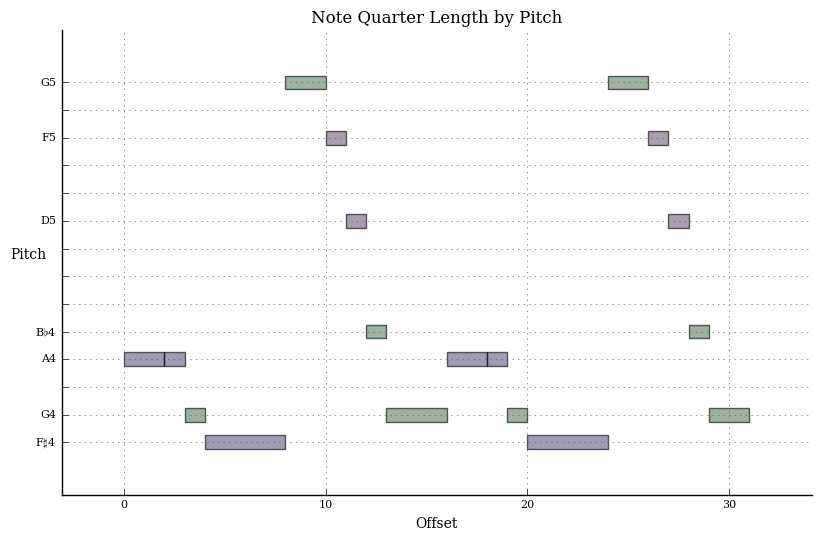


Figure 6 A Melody sequence expressed as Pitch vs time offset

### Sampling

MIDI notation captures musical information at a rate of 96 ticks [10] per beat. A beat is ¼-th of a note. So, MIDI uses (96 \* 4) ticks to represent one note. The information captured in raw MIDI files are of high resolution. This is too much information for this project, so I decided to use 8 ticks to represent 1 beat (1/4th of a note) which in turn means that 32 ticks will be used to represent one full note duration. The sequences in the dataset are 8 measures long. Each measure has 4 beats (1/4th of a note). So, there is a total of 32 beats in a sequence. A sampling rate of 8 ticks per beat produces an input sample of size 256(32 \* 8) samples for each sequence. Dataset contains 76 sequences of chords and melodies each. So, the total number of samples is 76 \* 256.

## Long Short Term Memory

An LSTM unit uses four gates to capture the long term dependencies namely forget gate, input gate, output gate and cell state. Each of these 4 gates have their own set of weights and in addition to that each LSTM unit also maintains a hidden state.

Forget gate( chooses which information to forget based on current input and hidden state information.

Input gate( chooses the information to be added to the cell state.

Cell state( is updated based on the information from the forget gate and input gate.

Output gate( controls the output from the neural network based on the current input from the cell state and current input.

Figure 5 shows the LSTM unit and how the different gates/states are computed.

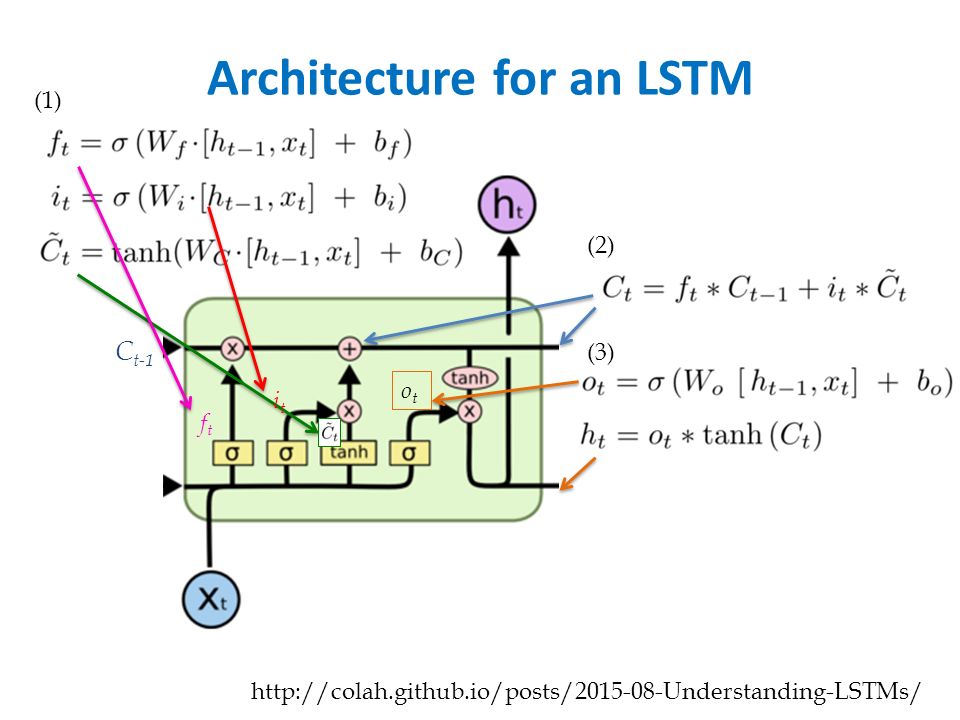


Figure 7 Long Short Term Memory

Multi-Layered LSTMs had the output of the LSTM in the previous layer fed as input to the next layer of LSTM units. An affine transformation was used in the final layer to produce output that gives the score for each class (piano note).

## Gated Recurrent Unit

Gated Recurrent Unit is an LSTM variant which uses 2 gates to capture the long term dependencies. Update gate and reset gates are what enables GRU to capture long term dependencies.

GRU activations are denoted by and it is a linear interpolation between the previous activation and candidate activation()

Update gate() decides how much the GRU unit updates its activation or content.

Reset gate() decides what part of the captured information can be reset.

Multi-Layered GRUs were implemented by feeding the output of the GRU unit in the previous layer and the final layer had an affine transformation to produce the score for each class (piano note).

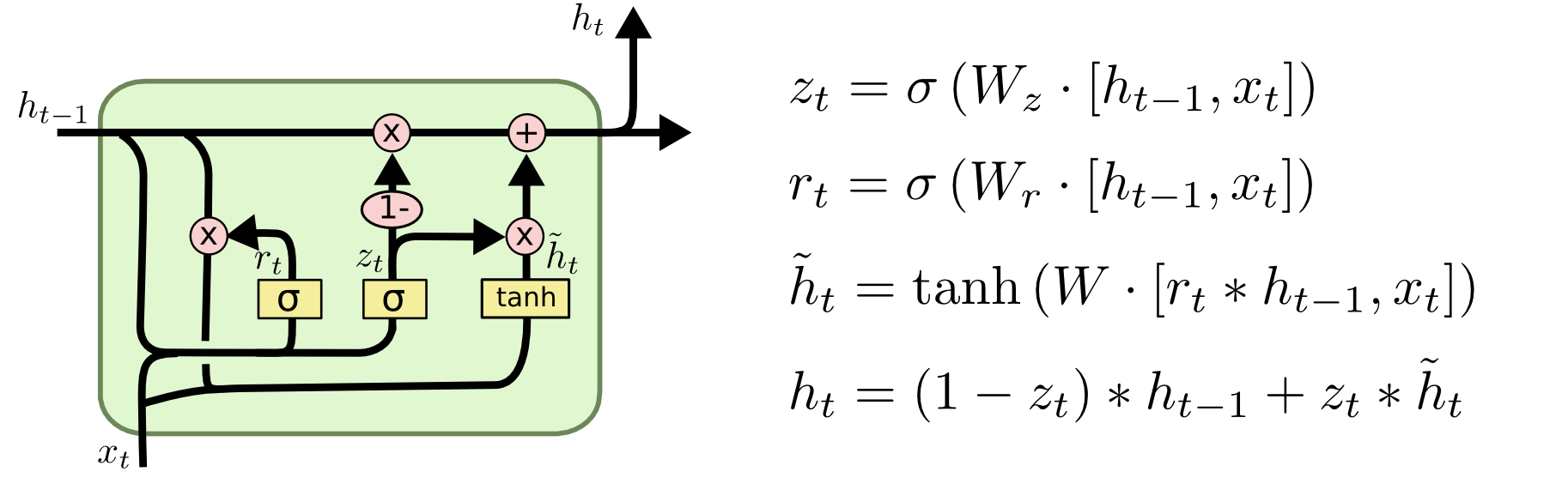


Figure 8 Gated Recurrent Unit

## Vanilla RNN

I trained a model using vanilla RNN to establish baseline values so that I can measure the performance of LSTM and GRU units.

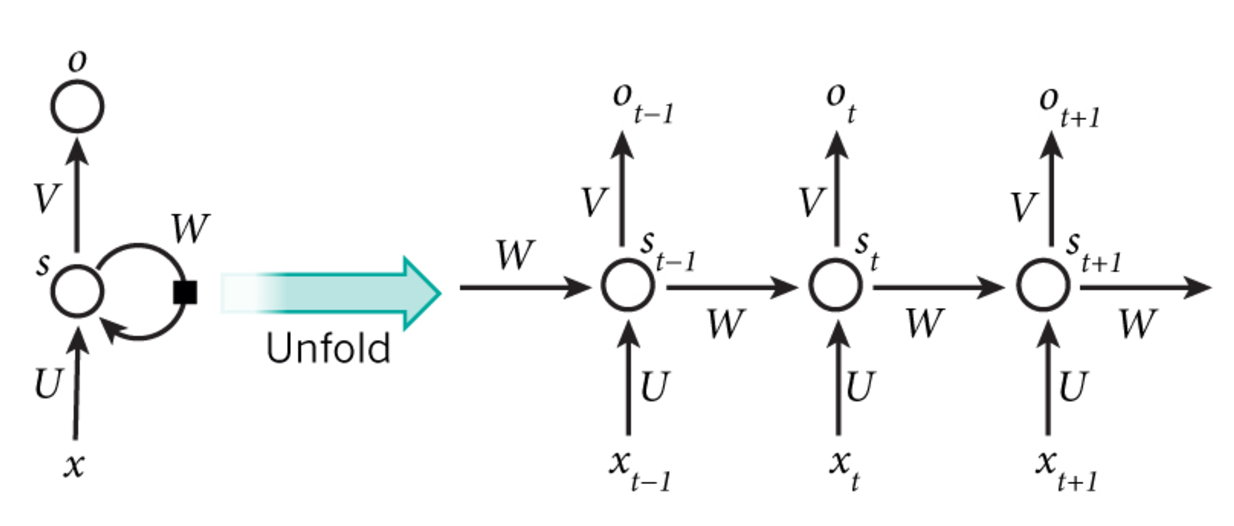


Figure 9 Vanilla RNN

Hidden state information is calculated as follows.

Output state is computed as follows

# Experiments

## Implementation Details

### Generating Train and Test samples

The dataset used for this project had 76 music compositions. Each music composition was sampled at 256 time steps giving rise to 256\*76 (19456) samples. 10% of these samples were chosen to be part of the test data and 5% of the samples were used as validation data set.

### Weight Initialization

One of the key factors that influence the performance of a neural network is weights initialization. In this project, the neural network parameters were initialized using Xavier initialization which was proposed by Xavier Glorot and Yoshua Bengio[6].

### Optimization

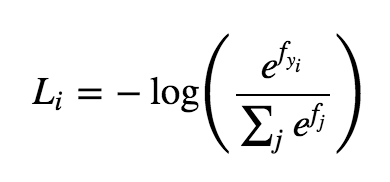
Weights and biases were updated after each iteration using “Adam” update algorithm. Batch size was chosen to be 128 for each iteration. A learning rate of 0.0003 was chosen after doing grid search over a range of learning rate values. A learning rate decay factor of 0.95 was applied after each epoch.

### Regularization

Dropout regularization was used to prevent the model from overfitting. A dropout probability of 0.2 was applied to the recurrent layers. Since the “active” notes are sparse, L1 regularization was applied to the weights shared by the recurrent layers.

### Loss Function

Softmax loss was chosen to measure the loss in each iteration because this is a problem of multi-class classification. We have to pick one piano-note from 25 possible piano-notes. Sigmoid activation was used in the output layer and the final loss were computed using Softmax.



. is the loss for input sample “i”. is the correct class for input sample “i” .

## Results

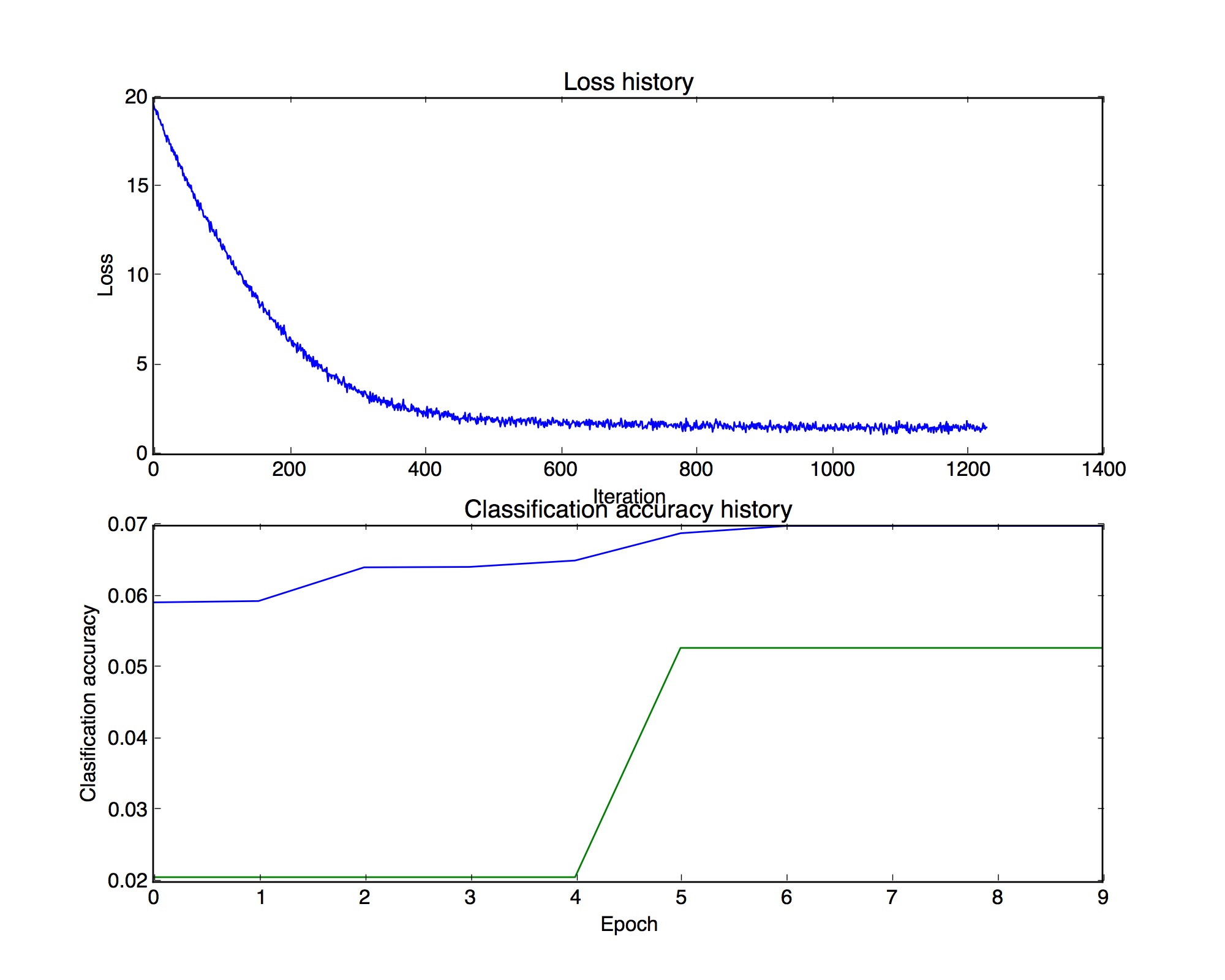
### Vanilla RNN

### GRU

### LSTM

It is hard to quantify what is aesthetically pleasing and what is not. For example, some people may find a musical composition pleasant while others may not like it so much. Having said that

#### 3 Layer LSTM Results



#### 4 Layer LSTM results

## 

#### 5 Layer LSTM results

## Evaluation

Evaluation criteria depends on the goal of the project. The goal of this project is to produce aesthetically pleasing melodies. The notion of an aesthetically pleasing to hear melodies is subjective. So, it is hard to quantify the “melodiousness” of a musical piece using science. So, I reached out to 5 human subjects and each of them were asked to rate the “aesthetically pleasantness” of the musical compositions. Each human subject was played 16 songs. Out of which 8 of them were generated musical compositions and 8 of them were human compositions.

# References

1. Good description of Piano-roll representation - http://www.bcp.psych.ualberta.ca/~mike/Pearl\_Street/Dictionary/contents/P/pianoroll.html
2. N.B Lewandowski, Y.Bengio and P.Vincent - Modeling Temporal dependencies in High dimensional sequences with application to polyphonic music generation - http://www-etud.iro.umontreal.ca/~boulanni/ICML2012.pdf
3. D.Eck and J.Schmidhuber – A first look at music composition using LSTM - http://people.idsia.ch/~juergen/blues/IDSIA-07-02.pdf.
4. Applying FFT to wav files - https://www.mathworks.com/matlabcentral/newsreader/view\_thread/164978
5. Polyphonic music - <https://en.wikipedia.org/wiki/Polyphony>
6. Xavier Glorot and Yoshua Bengio - Weight Initialization - <http://jmlr.org/proceedings/papers/v9/glorot10a/glorot10a.pdf>
7. Vanishing Gradient problem - <http://neuralnetworksanddeeplearning.com/chap5.html>
8. Music Notes - <https://en.wikipedia.org/wiki/List_of_musical_symbols#Notes_and_rests>
9. Notes and Rests - <http://www.tangomusicology.com/wordpres/music-rudiments/notes-and-rests-their-duration/>
10. Timing in MIDI notation -http://sites.uci.edu/camp2014/2014/05/19/timing-in-midi-files/
11. Understanding LSTMs – colah.github.io/posts/2015-08-Understanding-LSTMs/

1. In this report, the word RNN is used to refer to the class of Recurrent Neural Network architecture that includes but not limited to LSTM, GRU. The word vanilla RNN is used to refer to the basic recurrent neural network. [↑](#footnote-ref-1)