An approach to music generation with Recurrent Neural Network and Long Short Term Memory

Arthur Caron adcaron@kth.se
Florian Coissac coissac@kth.se
Côme Lassarat lassarat@kth.se
Victor Sanchez sanche@kth.se

Abstract

In this project, we explore the ability of a Recurrent Neural Network to generate classic music. The music is extracted from a dataset called MAESTRO which contains mainly MIDI audio files. To process these data, we focus on two approaches : The first one is from scratch and starts from the text generation process by doing what we called a Notematrix encoding on Matlab. The second one is based on an approach adapted audio signals properties with Tensorflow and Keras libraries on Python. Both approaches provide an explanation for the hyper parameters tunning.

1 Introduction

Deep learning and artificial intelligence allow the classification and regression of data such as images. This is typically done by feedforward neural networks, where the information is only handled in a single direction. However, temporal coherence sometimes exists in human speech, text, games, etc. Thus, the need to find techniques to emulate memory in neural networks emerged, to perform tasks like text translation, or speech recognition for example. Recurrent neural networks (or RNN) provide a solution to this challenge, thanks to the use of an internal state (memory) which allows the network to remember the sequence of previous inputs. Such a network can generate some text by training it on a Harry Potter book. The problem tackled in this paper is to train an RNN to generate music, another type of data with temporal coherence. Being able to reach this goal could prove to be useful for musicians to find inspiration, or to generate lots of royalty-free music for the internet creators to use as they wish. This will be done in two different ways:

- By encoding music into a sequence of characters, and using the code from Lab 4 to base our implementation of an LSTM.
- Using the TensorFlow Python library to generate directly the pitch, onset, and duration of notes.

In the TensorFlow approach, different configurations of layers, nodes, batch size, etc. will be tested to find the best one and see the influence of some parameters and techniques, such as dropout, on the learning process. And in the MatLab approach, experiments are conducted to try and measure the performance of the network's memory. **Link to the github repository**

2 Related work

Work on this subject can be found in scientific papers. The majority of it relies on one or several LSTM layers. The main difference between these approaches lies in the way to represent music. For example, in L. Johnston's work [1], the format used to encode a song is the ABC notation format. In D. Eck & J. Schmidhuber's work [2], a song is represented by switching on (output = 1) or off (output = 0) the notes individually during the duration of the latter. Time is represented by an input vector representing a slice of real-time. Another point of view is using 3 variables to represent a note when

Preprint. Under review.

training: the pitch, the step (time since the beginning of the song), and the duration (of each note) [3]. To optimize the training hyperparameters, some previous work studied dropout in 2014 [7], and the notion of temperature in 2015 [6].
Our TensorFlow approach is inspired by a tutorial called "Generate music with an RNN" made by TensorFlow [1], and the MatLab implementation of the LSTM forward and backward passes used the formula described in `http://arunmallya.github.io/writeups/nn/lstm/index.html#/` [8] (use the arrow at the bottom right to navigate in that web page).

3 Data

The dataset used in our work is the MAESTRO dataset (MIDI and Audio Edited for Synchronous Tracks and Organization) from `https://magenta.tensorflow.org/`, a branch of Google AI. It contains over 200 hours of piano songs in MIDI format. The package downloaded is composed of 1282 MIDI files.

4 Methods

4.1 Approach from Scratch - Networks and Solvers

In MatLab, we implemented the LSTM augmentation of the RNN from Lab 4, and we used a Xavier initialization for the network's parameters. The training was done using AdaGrad gradient descent for the RNN, and an Adam optimizer for the LSTM, both written from scratch as well. We also implemented a strategy using temperature when generating the music.

4.2 Approach from Scratch - Music encoding in MatLab

To manipulate MIDI files in MatLab we used the MIDI toolbox [4] which enables the transcription of a MIDI file into a note matrix containing every information for each note. We are using several other useful functions such as the piano roll which plots a note matrix in an easy-to-read way, or the playsound which can play the music represented by a note matrix. But to reuse the codes from Lab 4, we must feed the network a one-hot character encoding of the music dataset. Thus we designed our very own encoding that would give the network enough freedom to generate music as it pleases. The details of the note matrix encoding and character-wise encoding can be found in the Appendix A.1.

4.3 Comparing a generated song with a reference

To measure how close a generated melody is to the training set, we consider the notes as seen in a piano roll (see example in Figure [2]) as a cloud of dots. The space is three-dimensional with the pitch, onset, and duration of each note represented by a dot. To assess how similar two melodies are, we used the Chamfer distance (1), but before applying it we scaled each dot so that the pitch scale gets wider so that notes with different pitches will appear more distant. Then, to be able to compare measurements for various data sizes, we scale the result by dividing it by the Chamfer distance between the reference set and the zero origins.

$$Chamfer\_distance(X,Y) = \sum_{x \in X} \min_{y \in Y}(||y-x||_2) + \sum_{y \in Y} \min_{x \in X}(||x-y||_2) \tag{1}$$

4.4 Approach with Tensorflow

This approach is based on the previous work done on a tutorial by TensorFlow [3]. The goal is to output three characteristic parameters of the midi file data set. The **pitch**, which is the perceptual quality of the sound and is represented as a MIDI note number between 0 and 128.
The **step** is the time elapsed from the previous note or start of the track.
The **duration** is how long the note will be playing in seconds and is the difference between the note end and note start times.
The input is now a TensorFlow object, build based on a PrettyMidi [5]. transformation is done on the

same original midi file data set.
Instead of plotting the output space evolution over the epoch, we plot the final distribution of each estimated output. Then by comparison with the original input distributions, we can comment on the accuracy of the network. As the other indicator, we plot the piano roll which is the representation of each note played over time. It is a simple way to picture the sound without listening to it.

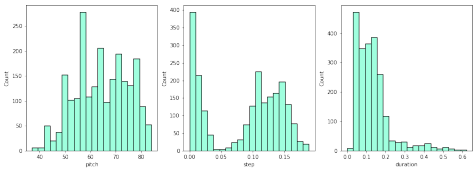

Figure 1: Example of initial pitch, step and duration distribution

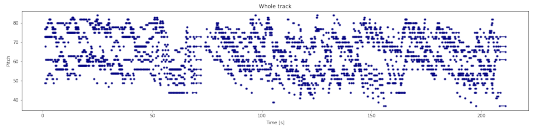

Figure 2: Piano roll over time

5 Experiment and Performances

5.1 Approach from Scratch

In our experiments we mainly changed the network's hidden state size d, the sequence length used during training seq_length, and the length of the training data $data\_size$. The other hyper-parameters were calculated as follows:

- learning_rate=1/d;
- n_epochs=round(sqrt(d)*1e4./data_sizes)*factor;

The multiplicative factor for the number of epochs needed was found through trial and error as it varied depending on the sequence length used in training. To check the convergence we verified plots such as the one showed in Figure 12 in the Appendix. We found it through trial and error for a smaller number of data sizes and hidden state lengths.

5.1.1 Memory performance analysis

To compare RNN and LSTM networks, and to measure the benefits of larger hidden states we would like to measure the maximum data size that a network can learn by heart. To determine that, we can train a network on various data sizes until convergence, and then make the network reconstruct the learned melody. If it succeeds then its memory can store data of that size, and thus we can find the limited capacity of a network's memory.
To have the network reconstruct the data it learned, we use a variant of the character generation from Lab 4 where we choose the character with the highest score in the network's output, therefore in this case we don't choose a character randomly based on the probability score.
A visualization of how well a network learnt some data is shown in Figure 3. The minimal loss shown in this Figure is the loss value at the end of training for a given network, with a training set composed of the $data\_size$ first characters. To make sure that the training did converge, we examine plots such as Figure 12 after the results are obtained to make sure everything went well.
Another visualization we made can be seen in Figure 4. These values are obtained by having the trained network generate a sequence using the deterministic synthesis. This sequence was then decoded into a note matrix to perform the Chamfer distance between the generated melody and the training set. The distance is finally divided by the Chamfer distance of the reference data with the origin, to normalize the results and make the values comparable through various data sizes.
Generating these results can take a few hours as each dot corresponds to complete training.

5.1.2 Memory performance results

After some experiments, we found that the best results obtained were those using a sequence length of 20, and the other hyperparameters used for each training are calculated using the formula above. It

is interesting to see that for a network with a given hidden state the loss values stay under a certain threshold until a certain data size. At that point, the network's hidden states aren't sufficient to memorize the data and thus the loss increases proportionally with the data size. We observe several things, the breakpoint where the network can't handle the data anymore is directly proportional to its number of hidden nodes. And it is way easier for the LSTM to deal with higher data sizes than for the RNN, for example, an LSTM with d=16 achieves a better result than an RNN with d=128. From these curves, we could read the maximum data size a network can learn as the data size where the minimal loss starts increasing. To check this affirmation we can consider the second metric: the Chamfer distance, which can be seen as the inverse of reconstruction accuracy.
The results obtained with the Chamfer distance are a lot messier and thus difficult to analyze. So we further investigated the results by looking at pianorolls of the generated and reference sequences. We found that an arbitrary threshold can be set at $1e-2$ below which the music is almost identically reproduced. Another threshold can be set at $1e0$ where the Chamfer distance between the two melodies is the same as if not a single note was generated. So when the results are even higher then it is really really bad (generally it is that the same note is repeated endlessly). The results are not as neat as the minimal loss curves, but a tendency shows that smaller networks and RNNs are faster at losing accuracy for smaller datasets than more complex networks.

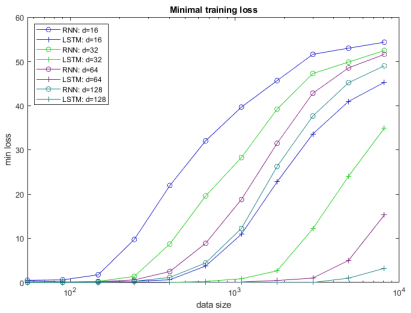

Figure 3: Semilog plots of the minimal training loss for RNN and LSTM, various hidden layer sizes and data sizes

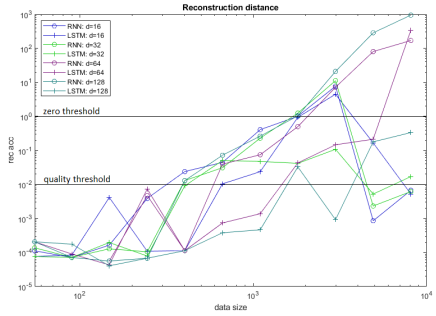

Figure 4: Log-log plot of the normalized Chamfer distance between the training set and generated melody for various hidden layer sizes and data sizes

5.1.3 Music generation

Though the deterministic generation can perfectly reconstruct a piece of music, see Figure 15 in the Appendix, we would like our network to generate with some creativity. Another loophole of synthesizing with a deterministic method is that it leads the network to create repetitive music, having the same cycle repeat itself during the whole generation. (See Appendix fig. 13) so it cannot be used for creative generation.
We then proceeded to generate music with a Temperature equal to 0.8 which was found to be optimal through trial and error. 14)

5.2 Approach with Tensorflow

The default parameter before the optimization phase will be taken as follow:

Sequence length	25	Batch Size	64	Learning rate	0.005	Nb of neurons	128
Data set size	1 file	Nb epochs	5	Type RNN	LSTM	Optimizer	Adam

Table 1: Default parameter before optimization

The data set contains classic songs which don't have the same rhythm at all. One can think that if we take several songs with the same rhythm, the network will be able then to generate several rhythms of

songs like with images. We analyze the memory of our model and we conclude that music are more complex than images so to avoid this complexity, we kept only one song by default. To compare the generated models we plot the distribution of each predicted variables and compare it to the same distribution for the original file.

5.2.1 RNN, LSTM or GRU cell

First, we have to choose whether we take a Simple RNN, a LSTM or a GRU cell. We train both models and we obtain the following results.

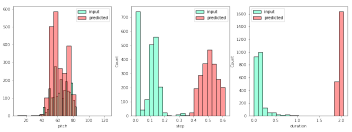

Figure 5: RNN

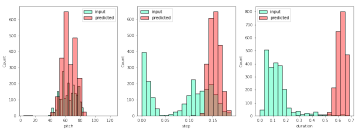

Figure 6: LSTM

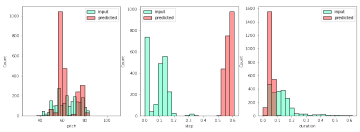

Figure 7: GRU

Here, the distributions (Pitch, Step, Duration) are not very well representative for all three models but we can observe that the LSTM perform better because their duration and step distribution are less local than with the RNN and GRU cell.
When we listen to the generated sound we can notice much more harmony and a better rhythm when we use the LSTM cell. In the appendix, you can observe the piano roll for both results.
Thus we will choose an LSTM cell.

5.2.2 Influence of sequence length and batch size

We decided to keep *batch size* > *sequence length* and we noticed that the learning was faster if these parameters were proportional. Thus we decided to take : $batchsize = sequencelenght * k (k \in Z)$.
The best results were obtained when k=8 and with a batch size of 64 notes and a sequence length of 8 notes.
We also try to keep k=8 but with a batch size of 128 and a sequence length of 16 notes but it appears that the frequency of note was much more important with the parameters and thus less representative of the original distribution.

5.2.3 Single hidden layer

The number of neurons on a layer has a major importance because it define how complex the neural network will be. Experiments showed that the best configuration for a 1-layer network is to have 128 nodes: it is better on the ear and the distributions of our 3 training variables are best represented (Figure 8)

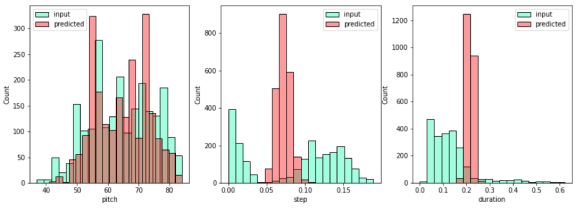

Figure 8: Distributions obtained with 1 hidden layer of 128 nodes

5.2.4 Multiple hidden layers, Dropout and Batch Normalization

The following experiments are made in the following configuration: $n_{epochs} = 10$, seq_length = 8 and batch_size = 64. The **influence of the number of layers** in the network is first analyzed. It can

be seen on Figure 9 that when the number of layers increases, at one point, the pitch distribution of the learned song and the generated one are almost identical. Then, the distribution looks less like the original one. Regarding the influence on the duration and step distributions, it doesn't seem to vary following a pattern. When adding **dropout** at each layer, nodes are randomly removed from the network, whose result can be seen on Figure 10: a random modification of the pitch distribution and the overall decrease of count of specific pitch can be observed. When comparing both songs (with and without dropout), it is possible to hear that the dropout song is composed of less notes. Regarding the impact of dropout on the duration and step distributions, it doesn't seem to vary following a pattern neither. What can be however noticed is that the step distribution is often squeezed toward the right or the left: the song is then either accelerated or slowed down, which is sometimes better or worse at the ear. Finally, **batch normalization** is added within each layer (with dropout). The pitch distribution seems concentrated on specific pitches. (Figure 11)

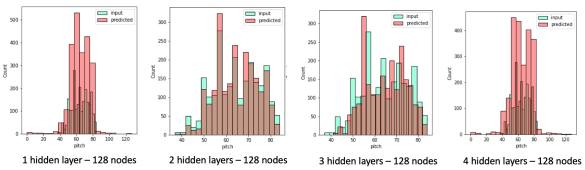

Figure 9: Pitch distribution according the number of hidden layers (of 128 nodes each)

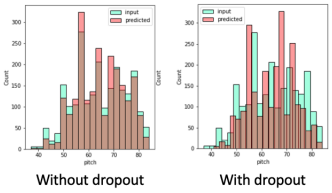

Figure 10: Pitch distribution obtained with and without dropout, for 2 hidden layers (of 128 nodes each).

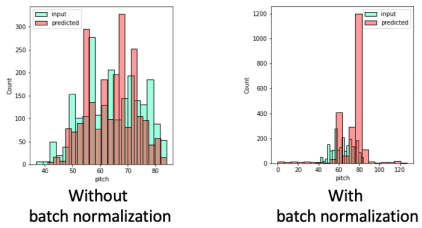

Figure 11: Pitch distribution obtained with and without batch normalization, for 2 hidden layers (of 128 nodes each).

5.2.5 Influence of Temperature

As the previous work by G. Hinton, O. Vinyals & J. Dean shown, the temperature parameter allow to have a softer probability distribution over classes. [6] When increase the temperature (see Appendix fig. 36), we can clearly observe a smoothing of the distribution. When we divide by a factor 10 (see Appendix fig. 35) thee distribution is quite restricted and barely represent one pitch. But when we multiply it by 10, then the distribution are more soft and wide. We find out that a good in-between is T = 0.8 . More notes are played and it is more harmonized.

6 Conclusion

The experiments conducted in this project showed that RNNs and LSTMs have a memory limit which increases for larger networks. This helps to optimize the size of the training data given to one of these networks. We showed qualitatively that a network which has full knowledge of a small melody can produce a different melody inspired from the original one. This could help artists who want to use a music they do not own, by letting them generate a very similar music that gives the feels the same as the original one, and that they would own entirely. It is yet hard to measure the quality of a music using maths, as the perceived quality of a music relies on the co dependence of the overall harmony of the pitch, and the grooviness of the rhythms. To go further in our work we wanted to compare the pros and cons of both music representations used in the two parts of the project.

7 References

References

[1] : L. Johnston (February 13, 2016) Using LSTM Recurrent Neural Networks for Music Generation https://drive.google.com/file/d/0B7oYxDkqYqqPYlR2LU01LVktbUU/view?resourcekey=0-s0Ft3PvUrWqTwSiroZaDkw

[2] : D. Eck & J. Schmidhuber (March 15, 2002) A First Look at Music Composition using LSTM Recurrent Neural Networks https://people.idsia.ch/~juergen/blues/IDSIA-07-02.pdf

[3] : (Last update: January 2022) Generate music with an RNN https://www.tensorflow.org/tutorials/audio/music_generation?hl=en

[4] : https://github.com/miditoolbox/

[5] : https://github.com/craffel/pretty-midi

[6] : G. Hinton, O. Vinyals & J. Dean (9 March 2015) - Distilling the Knowledge in a Neural Network - https://arxiv.org/abs/1503.02531

[7] : Srivastava, Nitish and Hinton, Geoffrey and Krizhevsky, Alex and Sutskever, Ilya and Salakhutdinov, Ruslan, 2014, Dropout: A Simple Way to Prevent Neural Networks from Overfitting

[8] : textitLSTM forward and Backward pass, http://arunmallya.github.io/writeups/nn/lstm/index.html#/, last consulted on 24/05/2022

A Appendix

A.1 Appendix Part - 4.1: Music encoding in Matlab

A.1.1 Notematrices

Notematrix is the data type used by the MIDI toolbox. It is a MatLab double array of size (n, 7) with n the number of notes played in the whole music. The seven columns each encode a note as follows:

ONSET	DURATION	channel	PITCH	velocity	ONSET	DURATION
(BEATS)	(BEATS)	(MIDI channel)	(MIDI pitch)	(or loudness)	(SEC)	(SEC)

Table 2: **Notematrix encoding**

The MIDI channel indexes different instruments in a track, and the velocity represents the loudness of the note but we do not play with these values in this work. The onset is the start time of the note and the duration its duration, expressed both in beats and seconds. And finally the pitch is the MIDI representation of the note's pitch, starting at 60 for a C4. The MIDI pitch encoding is well illustrated on this website https://newt.phys.unsw.edu.au/jw/notes.html.

A.1.2 Character-wise Encoding

Let's call X our double array representing the one-hot encoding of a song. The size of X is (K, n) with K the number of possible characters, and n the number of characters used to encode this song. The encoding is simple. A note of pitch p is encoded by the 'character' number p. The MIDI pitch starts at 21 so it leaves us the first 20 characters to work with and implement note duration, and time measures. We call the first character a separator. Its presence marks the beginning of the next semi beat. The other characters can be used after a pitch value to indicate the note's duration in semi beats. The tabular below summarizes the one-hot encoding.

Character number	1	2 to 20	21 to 108
Signification	Separator	Duration	Note pitch

Table 3: **Character-wise encoding**

A good example should help to grasp the encoding, let's consider the following sequence: [1 1 62 2 1 1 46 91 12]. The first two ones makes the skip from beat 0 to beat 1, where note 62 (D4)

starts for 2 semi beats. The next two ones make the skip to beat 2, where notes 46 (A2#) and 91 (G6) start, and they last respectively for 1 and 12 semi beats. The duration of 1 semi beat is implied by the absence of a longer duration.

A.1.3 Decoding

The decoding process requires a tempo and a velocity value to turn the character encoding back into a notematrix. Decoding must consider every "syntax error" which can't happen during encoding, but might show up in sequences generated by the RNN. For example, the sequence [1 5 1] should not exist because 5 should only be used as a note duration. We decided to consider it as a 1 instead. Then if a note is playing for 6 beats, but is said to start again only 2 semi beats later [60 6 1 1 60]. In that case we decide to ignore the second apparition of the note. Lastly we decided to translate a repetition of notes: [60 60 60 1] as a note duration: [60 3 1].

A.2 Appendix Part - 5.1

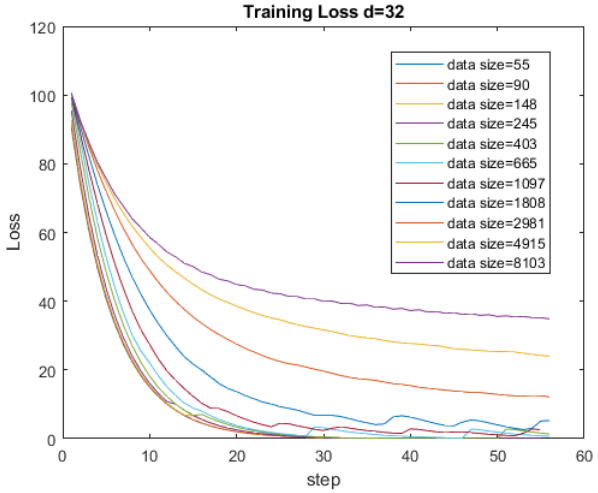

Figure 12: Example of a plot used to verify the convergence during experiments

A.3 Appendix Part - 5.1.1

Here is a piano roll of the first 400 notes generated by a deterministic method. The network used was a LSTN with 64 neurons and trained with a datasize equal to 664, which encodes about 400 notes. We observe the emergence of cycles in the music due to the deterministic generation.

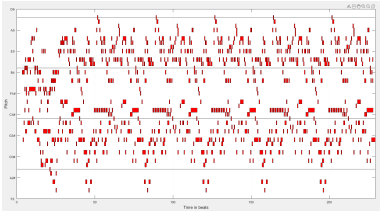

Figure 13: LSTM's piano roll with a deterministic generation, d=64

And in Figure 15 is a piano roll of the first 400 notes generated by a deterministic method with a 128 neurons LSTM on the same data size of 664. There the reconstruction is almost perfect. If we were to generate more characters, the melody would have looped and repeated itself endlessly.

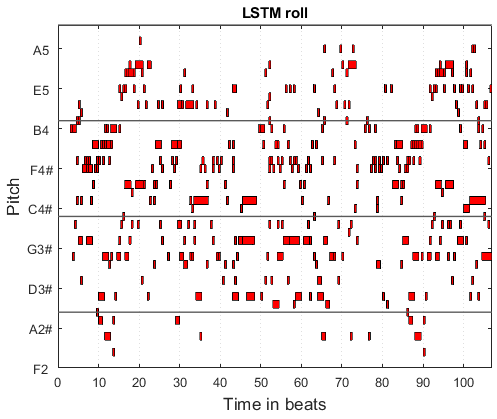

Figure 14: LSTM's piano roll with T=0.8

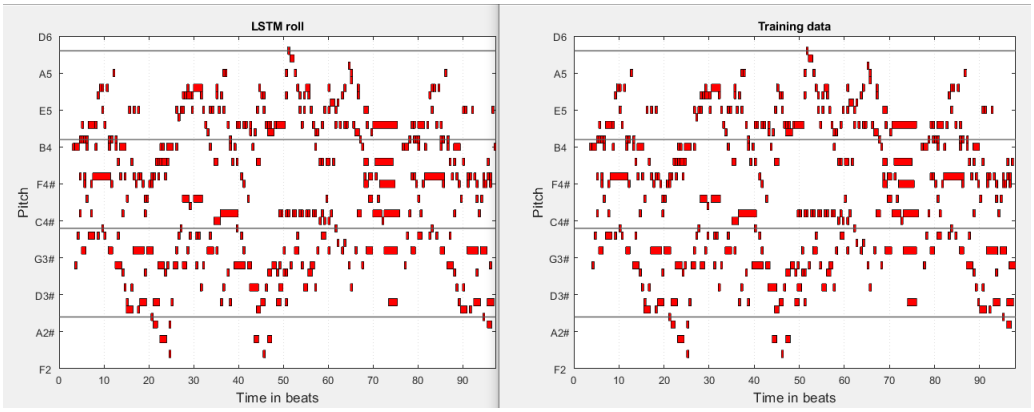

Figure 15: Almost perfect reconstruction using a 128 neurons LSTM and deterministic reconstruction on a 664 characters training sequence

A.4 Appendix Part - 5.2.1 - Piano Roll

Here are both piano roll as mentioned in the corresponding part.

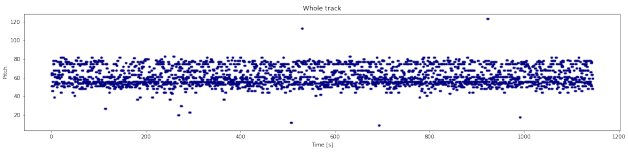

Figure 16: Piano Roll obtained with RNN Cell

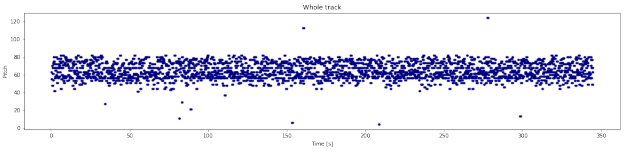

Figure 17: Piano Roll obtained with LSTM Cell

As mentioned before, during listening it seems that not every not is well represented on the audio generated by the RNN cell. We can observe that the piano roll of this audio has notes which form a thicker line in the bottom. So it is more concentrate on certain note. Which is matching with the impression during listening.

A.5 Appendix Part - 5.2.2

Here we kept a k = 8 but we modify the values for the batch size and the sequence length.

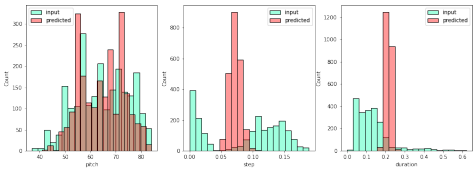

Figure 18: Distribution obtained with sequence length = 8 and batch size=64

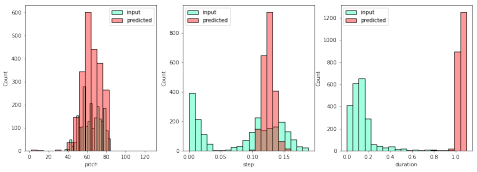

Figure 19: Distribution obtained with sequence length = 16 and batch size = 128

As mentioned in the part 5.2.2, we have a much better pitch distribution when the sequence length is made of 8 notes and the batch size is made of 64 notes. Knowing that the pitch is one the most important parameter because if some notes are too frequent and it is not harmonized, the final audio will be not that good.

A.6 Appendix Part - 5.2.3

Dropout method is useful when the complexity of the network is too important. It can be then interesting to dropout randomly some neurons during the training. Here the dropout has been done with a probability 30%. With a single LSTM hidden layer we did some dropout but it was not really relevant. In fact the duration and step distributions appeared to be completely changed from the original. (See fig. 21). By hearing the predicted sound, it is quite the same without dropout but the only difference is that some notes are missing which makes the sound worst. So Dropout technique is not really useful here.

The Batch Normalization (BN) helps to have more stability and more accuracy over the epoch. We implement it in our case and we observe that the distribution of the pitch is not really better. (See fig. 23) One interesting fact when we look at the piano roll generated, is that BN seems to focus on two main notes. (See fig. 24)

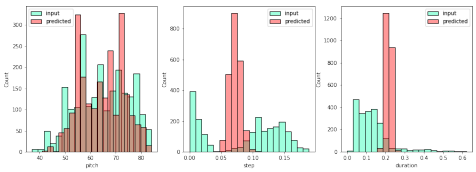

Figure 20: Distribution obtained without performing dropout

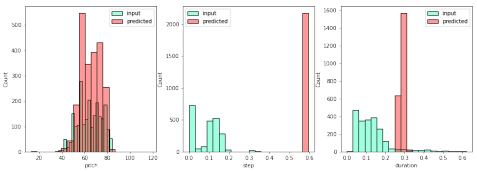

Figure 21: Distribution obtained with Dropout

For both result, we can observe on the above plots that the distribution does not match very well. In fact, the BN brings new notes which are not on the learnt midi file and which is not creating a good harmony. And the dropout makes the notes longer and forget about the step of a human.

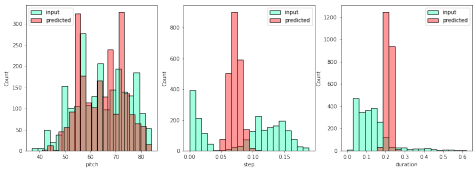

Figure 22: Distribution obtained with Unormalized Batches

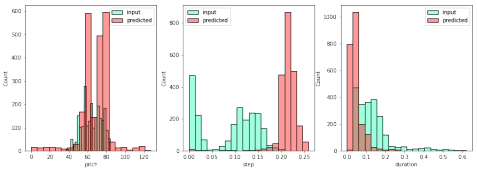

Figure 23: Distribution obtained with Normalized Batches

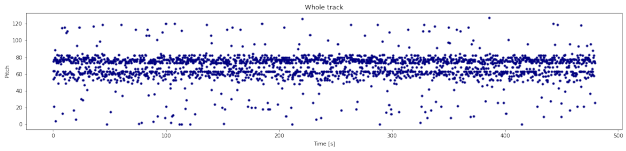

Figure 24: Piano roll obtained with Normalized Batches

A.7 Appendix Part - 5.2.4

The following experiments are made in the following configuration: $n_{epochs} = 10$, seq_length = 8 and batch_size = 64.

A.7.1 Influence of the number of layers

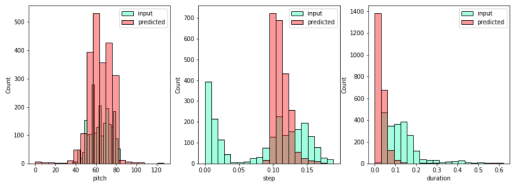

Figure 25: Distributions obtained for 1 hidden layer of 128 nodes

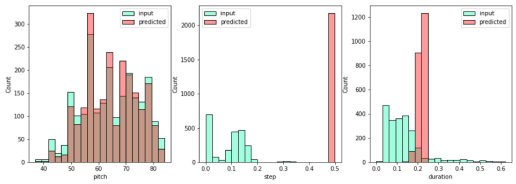

Figure 26: Distributions obtained for 2 hidden layers of 128 nodes

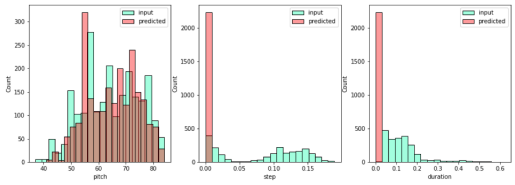

Figure 27: Distributions obtained for 3 hidden layers of 128 nodes

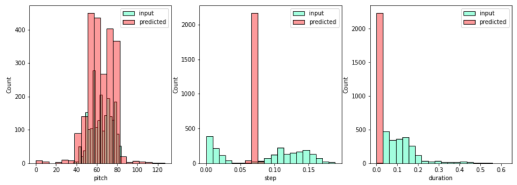

Figure 28: Distributions obtained for 4 hidden layers of 128 nodes

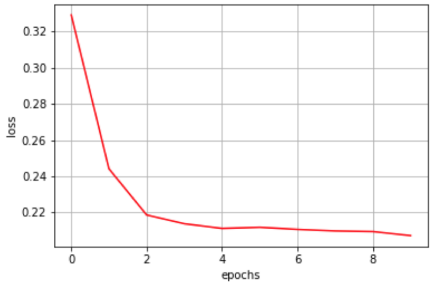

Figure 29: Loss function for a 2 hidden layer network of 128 nodes each

A.7.2 Influence of dropout

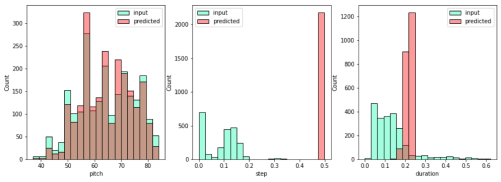

Figure 30: Distributions obtained for 2 hidden layers of 128 nodes without dropout

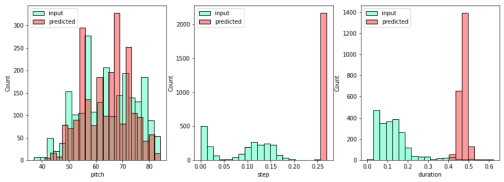

Figure 31: Distributions obtained for 2 hidden layers of 128 nodes with dropout

A.7.3 Influence of batch normalization

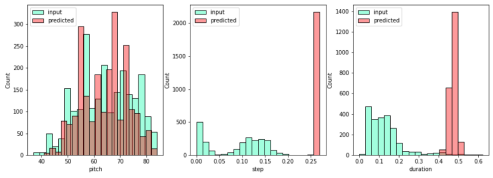

Figure 32: Distributions obtained for 2 hidden layers of 128 nodes with dropout and without batch normalization

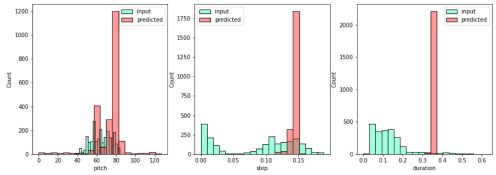

Figure 33: Distributions obtained for 2 hidden layers of 128 nodes with dropout and batch normalization

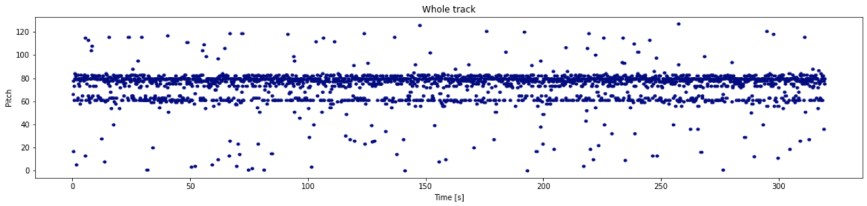

Figure 34: Piano roll obtained for 2 hidden layers of 128 nodes with dropout and batch normalization

A.8 Appendix Part - 5.2.5

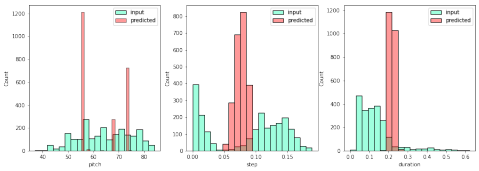

Figure 35: T = 0.1

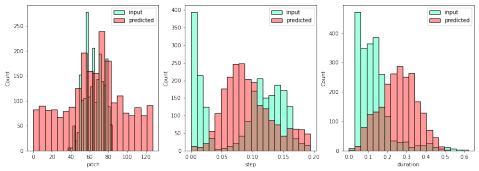

Figure 36: T = 10