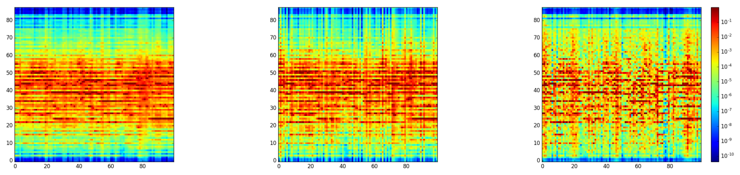
BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2017
http://excel.fit.vutbr.cz

Applying Recurrent Neural Network To Music Generation

Marek Majer*



Abstract
In this paper, we will teach computer to generate music sequences. We do this by modeling the probability of music sequences by a recurrent neural network and then sampling consecutive frames from the learned distribution. By further adjusting parameters of neural network we have managed to get 9.5 loss on validation data. Accuracy of the network was 0.17 with dropping out units at rate $p = 0.5$, which is in real usage higher, because of strict accuracy function. Work with neural networks led to creation of NeuralPiano, an application for desktops, that allows user to generate new music.

Keywords: Generating Music —Sequence modeling — Neural Networks — LSTM

Supplementary Material: Downloadable Code

*xmajer15@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

Music is one of the universal languages known to man. There were some artists who could understand that language a bit more than others. They could create beautiful and meaningful pieces. Could computer, one of the most stupid thing by itself, be able to understand what makes music beautiful and deliberate, if we just show it enough music data?

Computers nowadays are able to perform complex computations in a split second, which allows us to simulate some functions of human brain. We can then present artificial system with work of any artist, and the artificial system will slowly but steadily learn complex patterns in sequences

There is plenty of music editing softwares already available on market. Each of them offers many functions to adjust, rearrange, modify and anyhow change user's music. It's possible to delete parts of your music in editor in a no time, but there isn't a way to do the exact opposite, let the machine create music.

My aim si to create a program, which will not only allow the user to delete and adjust their music, but one that will also be able to recognize patterns in user's music and generate tones that will fit into others.

There has already been some papers about generating music using recurrent neural networks. The work of Boulanger-Lewandowski[1] was motivation for my work with neural networks and also provided datasets, that are used in this work.

2. Long Short Term Memory

Long Short-Term Memory (LSTM) is a special version of reccurent neural network with capabilities to store information for a longer time in a memory. It was introcuded by Hochreiter in 1997 [2]. LSTM in contrast to basic reccurent networks have special memory cell, which can store previous information very effectively, because of various gates (functions), that decide which information is essential and the way it is stored. I use LSTM that is defined by following:

$$\begin{aligned} i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\ \widetilde{C}_t &= tanh(W_c x_t + U_c h_{t-1} + b_c) \\ f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\ C_t &= i_t * \widetilde{C}_t + f_t * C_{t-1} \\ o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\ h_t &= o_t * tanh(C_t) \end{aligned} \tag{1}$$

Equations 1 desribe how a layer of memory cells is updated at every timestep t. In these equations :

- x_t is the input to the memory cell layer at time t
- $W_i, W_f, W_c, W_o, U_i, U_f, U_c, U_o$ and V_o are weight matrices
- b_i, b_f, b_c and b_o are bias vectors
- i_t is the input gate, f_t is the forget gate activation at time t
- C_t is the candidate value for the states of the memory cells at time t.

Last layer of neural network is called ouput layer. It generates ouput , which is set of numbers from interval $\langle 0,1 \rangle$. Every output of the output layer is then compared to desired output, difference from desired output is called error. Because more tones can be played simultaneously in music and are independent from others, error is calculated as cross-entropy (see eq. 2). Expected output t is compared to actual output y for whole output width n.

$$E_{total} = \prod_{x=0}^{x=n} (t_x y_x + (t_x - 1)(1 - y_x)) \tag{2}$$

To find the local minimum of error function, gradient descent algorithm is used, algorithm takes one steps proportional to the negative of the gradient. Because propagating error was slow, we also use optimization algorithm called back-propagation, which was proposed by Werbos [3]

3. Data

Datasets used in this paper are publiclz available at [1] They are already divided into three categories: training, validation and testing.

Music data in this work are represented as piano-roll. Which is an old way of representing piano-notes for self-playing pianos. Notes for piano are represented by their tones at discrete point in time. Representing notes as array of points in time is quite suitable for computers. Neural networks are working with vectors, therefore piano-roll is mapped into vector of length 88 (88 tones on piano).

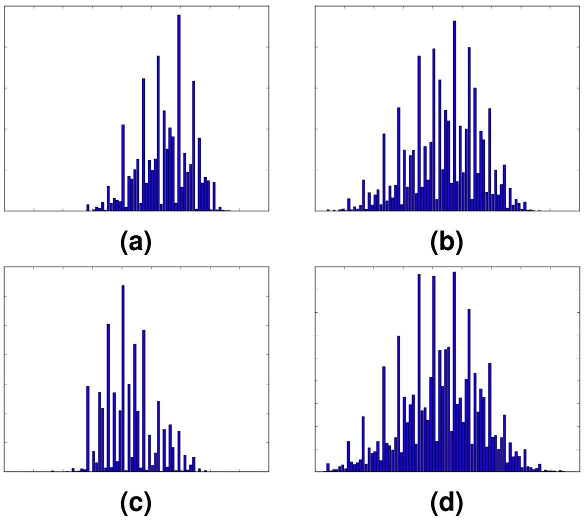


Figure 1. Quantity of tones in datasets: JSBChorales (a), MuseData (b), Nottingham (c), Piano-midi.de(d), It is notable that the majority of tones is clustered around the middle of the range. From the current dataset we could deduce that people dislike too many high or too many low notes. There seem to be some spaces between often played tones, those are the halftones, that are played less.

4. Optimizing the model

At the first step of creating neural networks, it is necessary to decide how many layers are needed for given problem. Adding more layers improve the power of neural network, although after certain point it does only make a little difference for huge increase in computing time.

Because the input and the ouput layer represent piano-roll at one distinctive time, they have to be set to length of 88.

In my case I have started with two hidden LSTM layers of width 1024 units. (See figure 2).

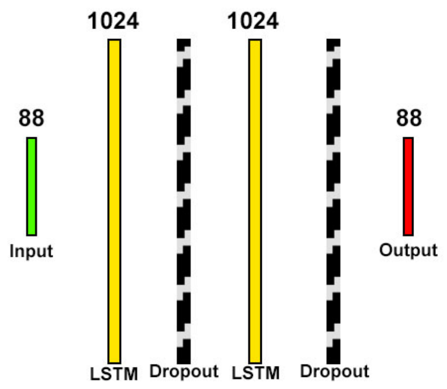


Figure 2. Neural network used in this work

[1]http://www-etud.iro.umontreal.ca/ boulanni/icml2012

To assest the quality of model, we use two metrics: loss function and accuracy function. A loss function specifies the goal of learning by mapping parameter settings to a scalar value specifying how bad those parameter settings are, therefore the goal of learning is to find a setting of the weights that minimizes the loss function.

Accuracy is the ratio of correctly generated output to destinated output, therefore we want to maximize the accuracy. Mine accuracy perceives correctly generated output only if all and only correct notes have been played. If 2 notes out of 3 are correct, acurracy of this scenario will be 0%.

4.1 Learning progress

Every going through entire training dataset is called an epoch. We can drastically help neural network in learning just by increasing the number of epochs. After the first epoch the neural network adjust only slighty. Some tones will get tiny chance of occurence and some will get huge chance of occurence. (See figure 3)

Adding more epochs only lineary lengthen the time of training, but after certain number of epochs, every new epoch will be less and less significant. Adding more epochs led to overfitting which will be addresed later in this paper. (See section dropout 4.2)

Epoch	loss	Validation accuracy	Validation loss
0	9.49914	0.15373	8.75719
5	8.14762	0.19001	8.83794
40	5.98347	0.16745	11.73323

Table 1. Validation accuracy and validation loss after certain number of epochs, validation accuracy was even higher after only five epoch compared to fourty, due to overfitting.

4.2 Dropout

When network learns the same set of data over and over, it can become to inclined to this data and lose its abillity to generalize. As a solution, dropout was proposed [4]. It works on priciple of randomly dropping out units in neural network. Each unit is dropped with a fixed probability p independent of other units.

p	Validation accuracy	Validation loss
0	0.15652	15.31470
0.1	0.16025	13.05284
0.2	0.16745	11.73324
0.3	0.14942	10.64415
0.4	0.15710	10.37890
0.5	0.170504	9.49869

Table 2. Different values of dropout in LSTM network after 40 epochs and 0.5 after 25 epochs.

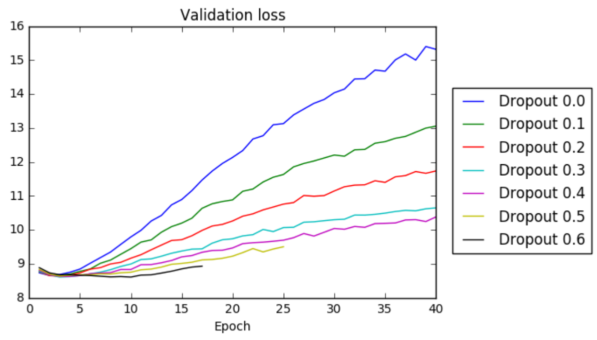


Figure 4. Loss of neural network depending on different values of dropout, with dropout 0.5 network crashed after 25 epochs

Dropout have proved to be an effective way to avoid overfitting for my case as well. (see figure 6) After increasing dropout even more, my network stopped working correctly, therefore I setled on dropout 0.5 at only 25 epochs.

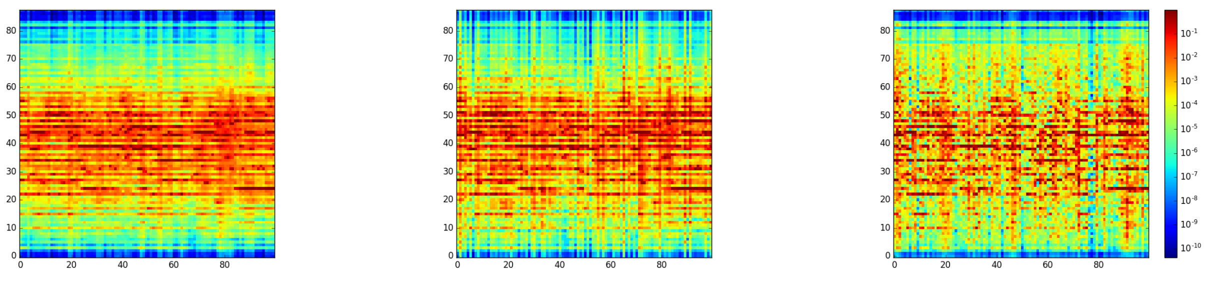
5. Generating music with neural network

A trained neural network can be presented with any note or a song to generate output according to input.

Output of neural network is a vector of length 88. Numbers in ouput are from interval $\langle 0, 1 \rangle$ Where every number is the probability of said tone to be played at the given time. Because in a music more tones can be played at the same time, the output of the neural network is sampled by generating array of random numbers of length 88. This array is then compared to ouput of neural network, if the random number is higher than output of nn, the tone is added to the note at that time. This sampling have proven to create quite listenable music.

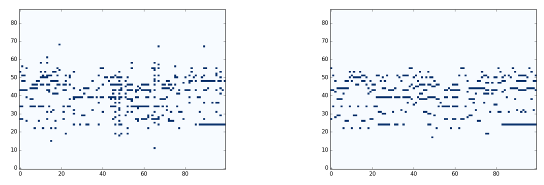
Since neural network was learned on classicial music, it is highly recomended to present initial sequences of the same genre. Output from network can then be sampled and used as new input. Doing this over and over whole song can be generated.

Competence of neural network can be also assest by generating notes from test dataset at any time t and comparing them to subsequent notes at next frame $t+1$(See figure 5)



(a) (b) (c)

Figure 3. Neural network output after one epoch (a), five epochs (b) and fourty epochs (c), we can clearly see that network learned which tones are played after first epoch, but by adding more epoch, network also learned when exactly are tones played, depending on the previously played notes



(a) **(b)**

Figure 5. Sampled neural network output (a) and actual output (b), some simialirities might be seen, mainly one note playing for several time units at the end. Also it seems like that neural network sometimes generate some notes that are not played at all and sometimes plays too many notes at once

6. Application - NeuralPiano

Based on my work with neural networsk, I have created compact application for generating music, called NeuralPiano.

NeuralPiano does not compete with any professional music editing software, but just provide new glance at possibility's in music editing.

NeuralPiano uses two LSTM layers of width 1024 units, with 0.5 dropout layers Neural network was trained only on 4 datasets of classicial piano music, therefore my application will be only able to recreate classical music.

It's compact and executable on any platform with python. Simple interface that looks like piano with timestamps allows user to easily create hiw own music, that can be instantly listened to. And above all allows the user to generate more music.

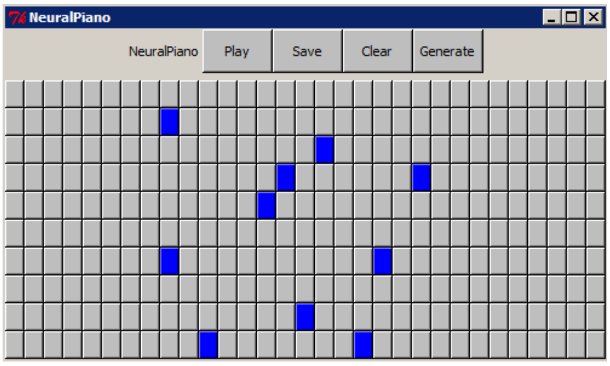


Figure 6. Alpha version of neural piano

7. Conclusions

With LSTM and enough time we have achieved even very 'human' things like creating listenable music.

For most common usage of LSTM, forty epoch of training was sufficient. We found dropout necessary to avoid overfitting, with optimal dropout probability $p = 0.4$, because higher value of dropout stopped the neural network before reaching forty epochs. Using dropout lead to result of

Small artists at home can now add something new into their pieces, thanks to NeuralPiano.

Other music editing softwares could highly benefit from implementing some kind of artificial inteligence to their software.Because generation with neural network is quite fast.

Acknowledgements

I would like to thank my supervisor Ing. Karel Beneš for his help.

References

[1] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. *Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription*, 2012.

[2] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[3] Paul John Werbos. *The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting*. Wiley-Interscience, New York, NY, USA, 1994.

[4] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.