Music generation using long short-term memory (LSTM) layers

Vasileios Kalogiras
KTH Royal Institute of Technology
vaskal@kth.se

Nikolaos Tatarakis
KTH Royal Institute of Technology
nta@kth.se

Abstract

In this project we try to synthesize music using recurrent neural networks when having as input raw music files. Specifically, we feed our network a music disc of *Gramatik* and expect to get in the end a sound that will sound similar to that. In this endeavour Long Short-Term Memory networks (LSTM) are used since their power of remembering past states is quite useful when working with time sequences. In the end, after training is done, we create 'music' sequences by providing our network with random seed points.

1 Introduction

Music production using Neural Networks has been explored in the past with a varying degree of success; Mozer in [1] used Recurrent Neural Networks (RNN) to synthesize music using a note-by-note approach but as he describes, "the compositions suffer from a lack of global coherence". Eck and Schmidhuber in [2] note that RNNs are not performing so well in this task mainly because of the well known issue of vanishing gradients. They address this issue by introducing Long Short-Term Memory layers (LSTM) and they also show, that, LSTM units succeeded in learning both global and local structure of the training data. They rely mostly on learning blues chords sequences in order to output a chord value.

More recently, the ability of Deep Neural Networks (DNN) has been investigated in the field of music generation. In [3], Greg Bickerman et al. shown that generative models such as Deep Belief Networks (DBN) can be used to produce Jazz melodies. They mapped their musical data into binary vectors, and, used this data representation in order to train a DBN based on multiple Restricted Boltzmann Machine (RBM) layers to produce a sequence of novel Jazz improvisations. Most of the aforementioned works use either notes or chords as the input material instead of raw music. This issue has been addressed by Nayebi and Vitelli in [4] as they successfully trained a recurrent LSTM deep network and managed produce music sequences using raw audio waveforms as inputs. They basically extracted frequency features from the raw music files using the Discrete Fourier Transformation (DFT) to obtain the desired feature vectors in order to feed the network with. The main concept here is to feed the network with Fourier feature vectors $X_1 \dots X_t$ where t represents time steps and let the network to predict the X_{t+1} feature vector, which will be part of the (desired) output sequence.

It is also worth noting, that, apart from music synthesis, DNNs have been also used lately in a wide variety of applications in order to generate every kind of things; In [5], Mordvintsev et al. discuss how to create visual art using the capabilities of Deep Convolutional Neural Networks (DCNN) while Karpathy in [6] discusses how RNNs with LSTM units are able to generate almost comprehensible and convincing texts ranging from image captions and Shakespeare play scripts to the extreme of algebraic geometry. In this paper, we will try to approach the problem of music generation using a deep architecture of LSTM layers influenced mostly by the works of [4] and [6] respectively.

The paper is organized as follows. In section 2, we describe the feature extraction method we used and we further review the deep LSTM framework as well as we introduce any relevant notation.

In section 3, the experimental process will be given in details along with information about the implementation. Experimental results are shown in 4 and we conclude in section 5.

2 Method

2.1 Feature Extraction

In order to use raw data for our experiments, we first need to find a reasonable way to represent it into an appropriate form so that the network can work with, such as matrices. Initially, we considered to extract MFCC coefficients from the training data in the same way as we did for the first lab, but eventually we turned down the idea as the MFCCs usually contain overlapping frames and maybe that could cause confusion to the network. Eventually, following the paradigm of [4] we extracted frequency information from the raw audio files using the Discrete Fourier Transform (DFT). More specifically, the audio files are split into N numbers of blocks and we run on each resulting block a typical DFT. This procedure results in two vectors that contain real and imaginary values, of N elements each. The final vector representation of each data block is a concatenation of these two resulting vectors and is basically used to train the network.

2.2 LSTM Architecture

As we mentioned earlier, LSTM units are supposed to learn long term-dependencies in the data, thus they are quite appealing for this task as we don't want our compositions to lack of global coherence as Mozer noted in [1]. LSTM layers are also able to address both issues of vanishing and exploding gradients by using the so called $memory\ cell$. Such a cell, cosists of four main parts; An $input\ gate$, a $forget\ gate$, $cell\ input$ activation vectors (also known as self-recurrent connection/connection to itself) and an $output\ gate$.

In this paper we use the LSTM architecture proposed by Graves in [7].

$$i_t = \sigma_i(x_t W_{xi} + h_{t-1} W_{hi} + w_{ci} \odot c_{t-1} + b_i) \tag{1}$$

$$f_t = \sigma_f(x_t W_{xf} + h_{t-1} W_{hf} + w_{cf} \odot c_{t-1} + b_f) \tag{2}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \sigma_c(x_t W_{xc} + h_{t-1} W_{hc} + b_c) \tag{3}$$

$$o_t = \sigma_o(x_t W_{xo} + h_{t-1} W_{ho} + w_{co} \odot c_t + b_o) \tag{4}$$

$$h_t = o_t \odot \sigma_h(c_t) \tag{5}$$

Here, i_t, f_t, c_t and o_t are respectively the $input\ gate$, $forget\ gate$, $cell\ input$ activation vectors and $output\ gate$. Hidden vector is denoted by letter h and has the same size as the aforementioned vectors. Where σ, denotes the logistic sigmoid function. W terms denote weight matrices (W_{xi} is the matrix of weights from the input gate to the input). The weight matrices from the cell to gate vectors such as w_{ci}, w_{cf} and w_{co} are diagonal for peephole connections. According to [8], the modern LSTM architecture contains peephole connections from its internal cells to the gates in the same cell to learn precise timing of the outputs. In our implementation we used a $tanh$ activation function for the feed-forward layer.

3 Experiments

3.1 Data

As training input we are using the disc *Coffee shop selection* of *Gramatik*[1]. The reason was that first of all, *Gramatik* distributes his music free of charge. Secondly, his songs don't contain lyrics which would add noise and make training more difficult.

[1] `http://www.gramatik.net/`

3.2 Parameters

Our whole project is based on implementations of the aforementioned architectures from *Lasagne* [9] which is a library that helps build and train neural networks in Theano [10]. Using *Lasagne* it is quite straight forward to resize or add layers. Due to time/allocation limitations however we didn't try many combinations. The structure of our main model had four LSTM layers with a feed-forward layer afterwards fixing the output size. Our hidden layer had 512 units and every batch had size 36. We used squared error as loss function and our weights were updated based on *RMSprop*.[2]

3.3 Music Generation

In order to generate a random sequence of music sound we start with a seed point, randomly chosen, from our training set, feed it to the network and get an output. Starting from there we begin building a sequence which we feed over and over to our network until we reach our preferred track length.

4 Results

4.1 Generated music

Some examples of generated 'music' can be found in this address[3] for different number of epochs.

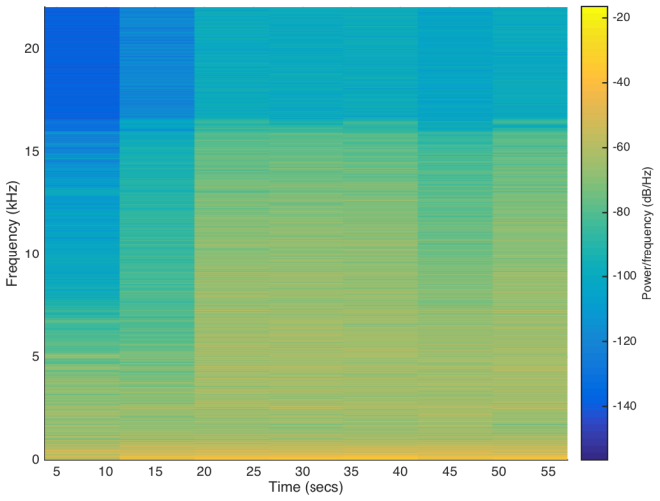
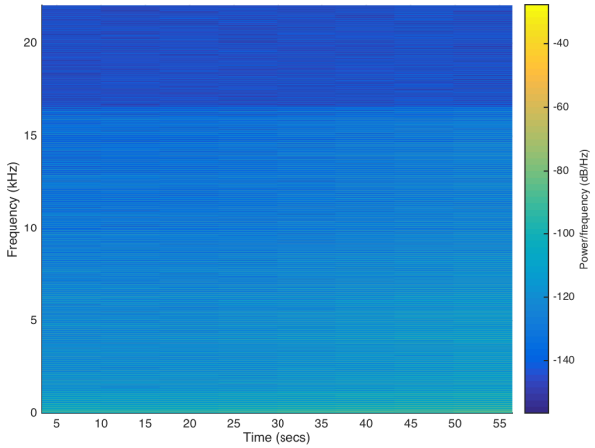


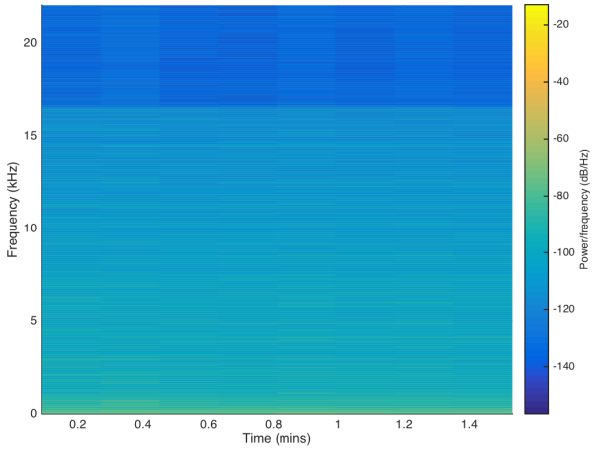
Figure 1: Spectrogram of a music file from training set

Here we can see some preliminary results of the system output. Figure 1 basically contains the ground truth spectrogram of a real sample from the training data, while, in Figure 2 we can see the spectrogram of the generated audio sequence after different amount of epochs. So far, from the obtained results we believe that the model cannot learn the training data well, which is also obvious from the sound output too. However, we didn't have the time to perform more tests. What is more, there is a chance that a bug exists somewhere in our code and didn't find it so far.

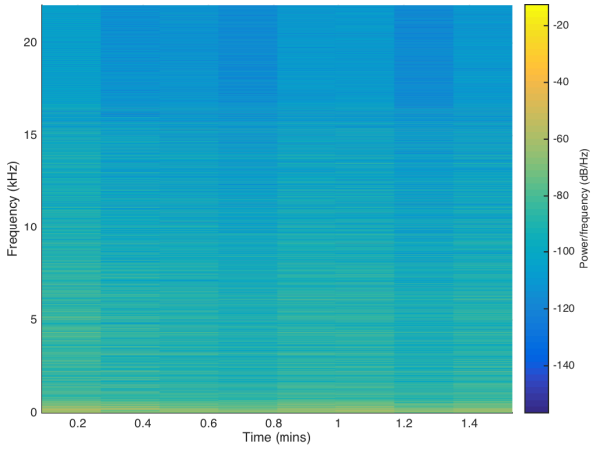
[2] `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`
[3] `https://www.dropbox.com/sh/mc8slqh1vo5aajq/AAAZKm_HqJfZS57n9zfZQz4Da?dl=0`



(a) Spectrogram of a generated music file after 100 epochs



(b) Spectrogram of a generated music file after 650 epochs



(c) Spectrogram of a generated music file after 950 epochs

Figure 2: Spectrogram changes over different number of epochs

5 Discussion and conclusions

In this project we examined the effectiveness of LSTM network layers in music synthesis from raw music inputs. From our results someone could claim the outcome wasn't successful at all. However, the idea seems to have lots of potential and we are certain there is room for improvement. As an extension or improvement we could have trained networks of different structures and see how that affects our outcome. Furthermore, instead of LSTMs someone can use the more recent Gated Recurrent Unit (GRU) [11]. However, the outcome should be expected to be similar, their main difference is that GRUs are less complex. What is more, even though we used music without lyrics trying the same thing having as input strictly electronic music, like *Skrillex*, for example could yield better results. Finally, trying different feature extractors instead of only discrete FFT could be interesting too.

References

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

[2] Douglas Eck and Jürgen Schmidhuber. Learning the long-term structure of the blues. In *Artificial Neural Networks—ICANN 2002*, pages 284–289. Springer, 2002.

[3] Greg Bickerman, Sam Bosley, Peter Swire, and Robert M. Keller. Learning to create jazz melodies using deep belief nets. In *First International Conference on Computational Creativity*, 2010.

[4] A. Nayebi and M. Vitelli. Gruv: Algorithmic music generation using recurrent neural networks. http://cs224d.stanford.edu/reports/NayebiAran.pdf, 2015. Online; accessed 22 May 2016.

[5] Christopher Olah Alexander Mordvintsev and Mike Tyka. Inceptionism: Going deeper into neural networks. Google Research, https://goo.gl/Bydofw, June 2015. Online; accessed 22 May 2016.

[6] Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. Personal Blog, http://karpathy.github.io/2015/05/21/rnn- effectiveness, May 2015. Online; accessed 22 May 2016.

[7] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[8] Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber. Learning precise timing with lstm recurrent networks. *The Journal of Machine Learning Research*, 3:115–143, 2003.

[9] Sander Dieleman, Jan Schlüter, Colin Raffel, Eben Olson, Søren Kaae Sønderby, Daniel Nouri, Daniel Maturana, Martin Thoma, Eric Battenberg, Jack Kelly, Jeffrey De Fauw, Michael Heilman, diogo149, Brian McFee, Hendrik Weideman, takacsg84, peterderivaz, Jon, instagibbs, Dr. Kashif Rasul, CongLiu, Britefury, and Jonas Degrave. Lasagne: First release., August 2015.

[10] Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, Alexander Belopolsky, Yoshua Bengio, Arnaud Bergeron, James Bergstra, Valentin Bisson, Josh Bleecher Snyder, Nicolas Bouchard, Nicolas Boulanger-Lewandowski, Xavier Bouthillier, Alexandre de Brébisson, Olivier Breuleux, Pierre-Luc Carrier, Kyunghyun Cho, Jan Chorowski, Paul Christiano, Tim Cooijmans, Marc-Alexandre Côté, Myriam Côté, Aaron Courville, Yann N. Dauphin, Olivier Delalleau, Julien Demouth, Guillaume Desjardins, Sander Dieleman, Laurent Dinh, Mélanie Ducoffe, Vincent Dumoulin, Samira Ebrahimi Kahou, Dumitru Erhan, Ziye Fan, Orhan Firat, Mathieu Germain, Xavier Glorot, Ian Goodfellow, Matt Graham, Caglar Gulcehre, Philippe Hamel, Iban Harlouchet, Jean-Philippe Heng, Balázs Hidasi, Sina Honari, Arjun Jain, Sébastien Jean, Kai Jia, Mikhail Korobov, Vivek Kulkarni, Alex Lamb, Pascal Lamblin, Eric Larsen, César Laurent, Sean Lee, Simon Lefrancois, Simon Lemieux, Nicholas Léonard, Zhouhan Lin, Jesse A. Livezey, Cory Lorenz, Jeremiah Lowin, Qianli Ma, Pierre-Antoine Manzagol,

Olivier Mastropietro, Robert T. McGibbon, Roland Memisevic, Bart van Merriënboer, Vincent Michalski, Mehdi Mirza, Alberto Orlandi, Christopher Pal, Razvan Pascanu, Mohammad Pezeshki, Colin Raffel, Daniel Renshaw, Matthew Rocklin, Adriana Romero, Markus Roth, Peter Sadowski, John Salvatier, François Savard, Jan Schlüter, John Schulman, Gabriel Schwartz, Iulian Vlad Serban, Dmitriy Serdyuk, Samira Shabanian, Étienne Simon, Sigurd Spieckermann, S. Ramana Subramanyam, Jakub Sygnowski, Jérémie Tanguay, Gijs van Tulder, Joseph Turian, Sebastian Urban, Pascal Vincent, Francesco Visin, Harm de Vries, David Warde-Farley, Dustin J. Webb, Matthew Willson, Kelvin Xu, Lijun Xue, Li Yao, Saizheng Zhang, and Ying Zhang. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[11] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Gated feedback recurrent neural networks. *CoRR*, abs/1502.02367, 2015.