Learning the decay of a sound signal

Jörg Sander
jorg.sander@toologic.com

Frederik Harder
frederik.harder@student.uva.nl

Abstract

This project trained recurrent neural nets (RNN) on single note raw audio signals in order to investigate their capacity for generating tones with natural decay. Unlike in much of the current work in audio generation with RNNs, the goal is not to create novel output, but clean and accurate reconstructions.

1 Introduction

Recurrent neural networks have attracted a number of researchers in the past who attempted to use these models for the generation i.e. composition of music. Generating audio signals directly rather than in midi or abc-notation is a fairly new field of research, however. While first attempts show that it is possible to generate novel audio sequences based on training data with fairly simple models, these results often contain high levels of noise and a complex jumble of features from the training data. In this project we aim to use a simpler more controlled setup to produce a clean reconstruction of simple instrument samples consisting of individual notes.

1.1 Fundamentals

An RNN can model a probability distribution over sequences, i.e. $p(x_{1:T})$

$$p(x_{1:T}) = \prod_{t=0}^{T-1} p(x_{t+1}|x_{1:t})$$

Each $p(x_{t+1}|x_{1:t})$ is represented by the output of an RNN at a single time step, identifying each of its components with the statistics of the distribution (e.g. Bernoulli or multinomial logistic regression aka softmax regression). The representational power of an RNN is mainly determined by the output function f_θ that maps the RNN hidden state $\mathbf{h}_{t-1}$ to a probability distribution of possible output, where θ is the set of parameters of f.

Training an RNN has been hard because of the *vanishing gradient* problem (Hochreiter & Schmidhuber, 1997, [1]). Methods such as Back-Propagation Through Time (BPTT) compute the gradient of the loss function and update the network weights by propagating errors *back in time*. As the error propagates from layer to layer, it tends to either explode or shrink exponentially depending on the magnitude of the weights. RNNs that need to predict long time sequences can be seen as deep networks (unrolled over time). In both cases the network fails to learn long-term dependency between inputs and outputs which made them especially unsuited for modelling musical sequences because these usually span across multiple notes in time.

Hochreiter & Schmidhuber (1997) [2] designed the Long Short Term Memory (LSTM) algorithm that solved the *error flow* problem and therefore these recurrent networks are nowadays commonly used for modelling long term dependencies in generic sequence modelling tasks. Recently, the Gated Recurrent Units (GRU) were introduced by Cho et al. (2014) [8] which can be mainly seen as a simplification of the LSTM model.

A Convolutional Neural Network (CNN) is comprised of one or more convolutional layers (often with a subsampling step) and then followed by one or more fully connected layers as in a standard

project AI MasterMusic (2016), generative audio , University of Amsterdam.

multilayer neural network. The architecture of CNNs as used in Computer Vision is designed to take advantage of the 2D structure of an input image, by extracting local features independent of their position in the image, but this method can also be adapted to other data, which has localized features along one or more dimensions. such as a speech signal or raw audio signal. This is achieved with local connections and tied weights followed by some form of pooling which results in translation invariant features. Another benefit of CNNs is that they have fewer parameters and are thus easier to train than fully connected networks with the same number of hidden units.

In this paper we present the results of modelling the natural decay of raw audio signals by means of a simple LSTM model and a more sophisticated model that consists of a convolutional neural network that acts as an encoder for the input to an LSTM model. To our knowledge the second model (which we denote CNN-LSTM) has never been tested before on raw audio signals.

2 Related work

Eck & Schmidhuber (2002) [2] were the first that employed LSTM models to learn and compose blues music. Without being complete, Franklin (2006) [3], Boulanger-Lewandowski et al. (2012) [4], Liu & Ramakrishnan (2014) [5], Bayer & Osendorfer (2015) [6] and Fabius & van Amersfoort (2015) [7] used increasingly sophisticated combinations of RNNs with Deep Bayesian Networks (DBN) to model audio sequences. All these models were trained on binary vectors to represent individual notes. These representations can be classified as low dimensional compared to the raw audio signal (e.g. in the well known *wav* audio format a second is encoded by 44100 integer values). Our models are trained directly on raw audio signals thus giving the generative models the chance to access the entire audible frequency spectrum.

Recently two Stanford students Nayebi & Vitelli [9] conducted a project called GRUV in which they trained an LSTM and GRUV model on raw audio signals and used these models to *improvise* new music. Another project by Jakub Fiala [11] extends the work of Vitelli & Nayebi to audio voice signals and kick-drum samples.

Most recently Chung et al (2016) [10] have used a combination of a Variational Auto-Encoder (VAE) and RNN (called VRNN) to model high dimensional raw audio speech signals. Their approach is very promising and is definitely giving a useful direction into future research work. For now, however, we choose a simple RNN model and a combination of RNN and CNN to conduct our experiments.

3 Model

LSTM layers can be embedded in different neural net architectures and usually some feed-forward layers are added before and after the recurrent layers. In this project we tried several configurations, which can be grouped into the two classes discussed below.

3.1 LSTM with dense layers

Following the example of Nayebi and Vitelli [9], we chose to start with a simple architecture consisting of an LSTM layer surrounded by two dense layers. Models of this type are varied in the number of units per layer, activation functions, and the number of recurrent layers. This architecture is about as simple as it gets for recurrent networks and we expected that any performance achieved here should be reproducible and improved on by more elaborate configurations.

We fix the model initially to a single recurrent layer with 1024 LSTM units, with the number of units in the input and output dense layers always equal to that in the LSTM layer. At first the dense layer units used a linear activation function opposed to the LSTM with a $\tanh$ function. We also investigate the model performance using the $\tanh$ and relu function as activations for the dense units. Naturally we vary the number of LSTM units using values between 256 and 2048. We also conduct experiments in which the model contains two recurrent layers. For the final paramenter, we train the model with varying time resolutions, specifically 10, 30, 60, 90 and 120 time slices per second.

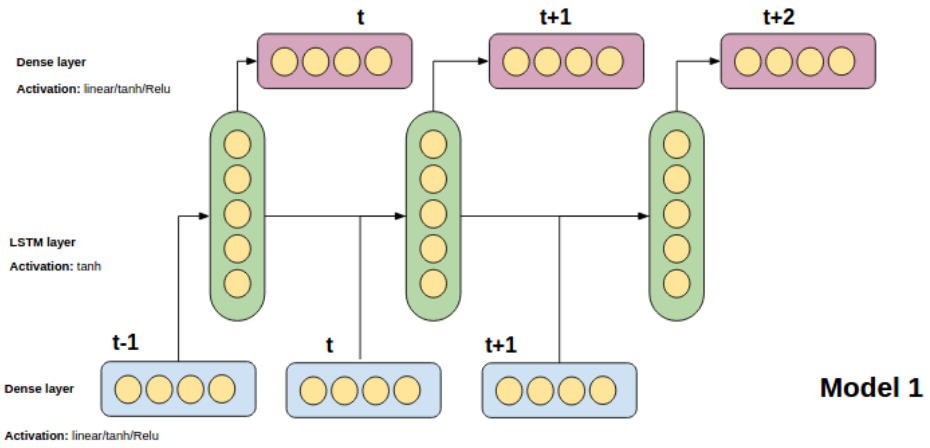

Figure 1: Model 1: LSTM with dense layers

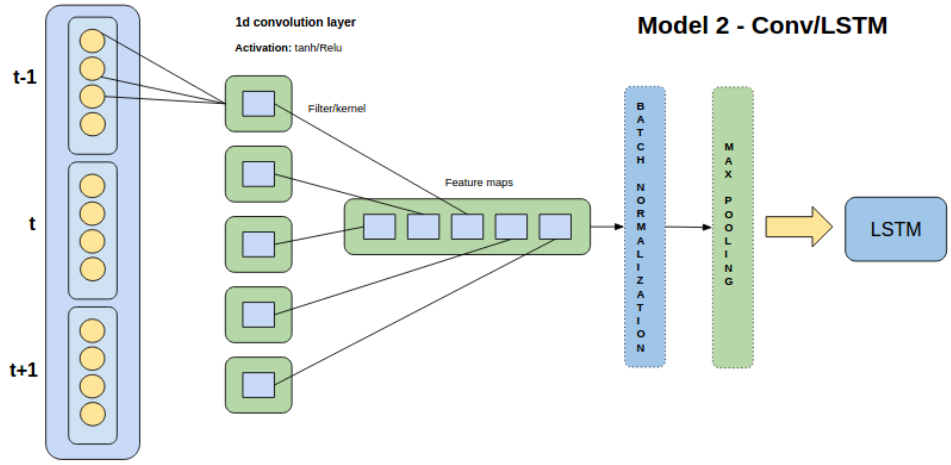

Figure 2: Model 2: CNN-LSTM

3.2 LSTM with convolutional layers

The architectures of the second model are based on the fixed best configuration of the recurrent neural network (based on the results obtained with the first model), with the exception of the input dense layer (to the LSTM) which we optionally remove or keep. We start experimenting with a single one-dimensional convolutional layer, only varying the activation function (relu, tanh, sigmoid, linear), the number of filters (8, 16, 32, 64) and the filter length (in a range between 3 to 51). The border mode of the filter is fixed to *same*. We then extend the model in successive steps up to three one-dimensional convolutional layers, separating the conv layers by a batch-normalization and one-dimensional max-pooling layer (using stride 2 and setting border mode to *valid*). Depending on the number of filters used in the first layer, the successive convolutional layers always contain significantly fewer filters, roughly half of the size of the previous layer and never go lower than four. The equivalent applies to the filter length, which decreases with the second and third convolutional layer.

4 Experiments

Both models are implemented by means of the *Keras* framework which allows for easy and fast prototyping. Theano is used as backend system. The experiments were run in part on a private Linux

Layer type	Initialization function	Remark
Dense	Glorot uniform	
LSTM	Glorot uniform	Orthogonal for inner layers of LSTM unit
1D-Convolutional	Uniform	

Table 1: Initialization function for layer weights

system with a modern GPU and in part on the DAS4[1] supercomputer. The implementation of the experiments is available on github [2].

Data-sets We evaluated both models on four different data-sets that we obtained from the *Electronic Music Studios* of the University of Iowa [3]. All samples are studio recordings of individual notes:

1. **Flute**, 39 samples each approximately 3 seconds long. Maximum sampling frequency $\approx$ 2000 Hz;
2. **Guitar**, 45 samples each approximately 5 seconds long. This was the most homogeneous data-set, containing 26 C-notes. Maximum sampling frequency $\approx$ 1400 Hz;
3. **Cello** (bow and plucking), 100 samples each approximately 3 seconds long. Maximum sampling frequency $\approx$ 8000 Hz;
4. **Piano**, 88 samples each approximately 5 seconds long. Maximum sampling frequency $\approx$ 4200 Hz.

Preprocessing and training The different data-sets are downsampled according to the *Nyquist Shannon* sampling theorem. Having to select a maximum sampling frequency for each instrument, we opt for the highest pitch which can be produced on the instrument[4]. After downsampling, the signals were scaled to values between -1 and 1 and each data-set was normalized using the global mean and standard deviation computed from the entire instrument training set. These statistics are saved and used during the generation of new signal sequences. Each signal is then divided into time slices, the length of which is denoted as *block size* and is calculated based on the new sample rate and the *temporal resolution* (chunks per second) which is a parameter of our experiments. Fixing the time resolution, block sizes can be different for the various instruments due to the specific sampling rate of a musical instrument.

Each model is trained with gradient descent using the RMSprop [5] optimizer. The loss is calculated by means of the mean squared error (MSE). All data-sets are trained for 180 epochs and a mini-batch size of 10. After training each model is saved together with the trained weight matrices. Table 4 shows the weight initialization functions that are used for the different layer types of the models.

Generating decay sequences The trained models are tested mostly on the training set. The best model that we obtained for the guitar data-set was also tested on *unseen* audio signals of the same instrument. A model is primed initially with approximately 330 ms of the original sound signal and successively predicts a new signal sequence that is equally long as the prime signal, shifted by one time slice. The last time slice of the generated signal is then appended to the prime signal causing the model to generate increasingly longer signal sequences. Figure 3 illustrates the generation process for three consecutive iterations. The final sequence generated by the model has the same length as the original signal. In order to evaluate the results, we store a) the original signal b) the generated signal which also includes the prime and c) the generated sequence without the prime.

4.1 Evaluation criteria

In order to determine the quality of our generated signals we used a number of qualitative measures, which can be judged quickly. Given that the initial results differed significantly and the rate of

[1] `http://www.cs.vu.nl/das4/`

[2] `http://github.com/Fr-d-rik/generative_audio`

[3] `http://theremin.music.uiowa.edu/MIS.html`

[4] `http://www.zytrax.com/tech/audio/audio.html#frequencies`

[5] RMSprop is an unpublished, adaptive learning rate method proposed by Geoffrey Hinton

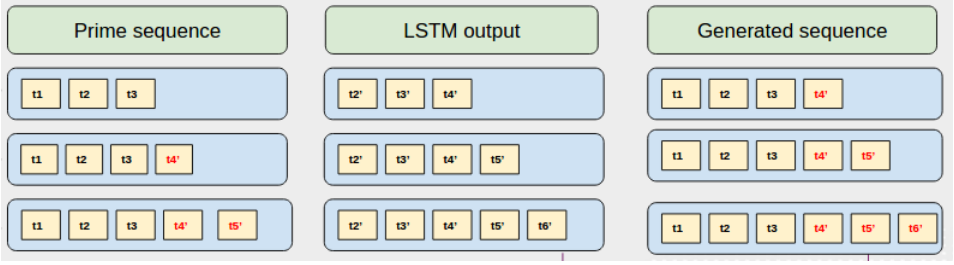

Figure 3: Sequence generating process

generation was slow enough for manual checking, this was more effective than designing quantitative measures right from the start.

To have a decaying signal, three requirements must be met:

1. The signal must maintain the same dominant frequencies
2. The signal must decrease
3. The level of (audible) noise introduced by the RNN must be sufficiently low.

Requirements one and two are straightforward and can be evaluated by listening and by looking at a short-time Fourier transform of the signal (STFT)[6]. The third one is arguably a quantitative measure and *sufficiently low* is of course very vague. In practice, however, the noise levels in our generated signals differ so greatly, that a rough categorization and comparison between signals can be done by hand. In many setups the noise level becomes as loud as the signal. For others it is only audible over headphones or in a fairly quiet environment, the latter of which we would, for now, classify as sufficiently low.

As a second step, signals meeting the three base requirements should also be evaluated on quantitative measurements. Given our current results, this step is still a bit further off, but we have implemented a function for estimating the decay envelope on signals, modelled with simple exponential decay. The parameters of this envelope function serve as a measure of how closely a generated decay curve resembles that of a signal. In addition we propose an analysis on the signals STFT representation. In the frequency dimension, a number of extrema are selected (the location of which should be largely static over time in our samples) and decay curves are fitted for only these dominant frequencies. The aim of this measure is to exclude the impact of noise on the estimated decay, as only frequencies from the instrument are taken into account. At this stage noise must also be taken as a separate quantitative measure.

4.2 Results

After having tested varying setups for the LSTM with dense layers, we find that the majority models consistently manage to reproduce the prime signal for a short time on the cello and guitar data-sets. The level of noise is generally quite high and often becomes louder than the prime signal, while decay times vary from about a tenth of a second to no decay at all. Our best results on each data-set were obtained with the number of units in each layer at 512, a single recurrent layer and choosing a temporal resolution of 60 time slices per second. Greater number of units and fewer - and thus larger - time slices seem to give too much freedom to the model, while fewer units and smaller time slices quickly reduce the modelling capacity to a point, where it becomes incapable of modelling the signal. Adding one additional recurrent layer seems to have little effect on the model. At three recurrent layers, the added complexity starts to increase noise, while each layer is still sensitive to a reduction of units. Adding spectral information in form of the Fourier transformed data representation to the model seems to have only adverse effects as well. If the model does make use of the information contained in the added representation, it is not enough to compensate for the added model complexity introduced by doubling the input size. The data-sets, as quickly became apparent, vary substantially in their degree of difficulty and will be discussed separately.

[6]STFT spectrograms displayed in this report display the square root of the actual values in order to highlight noise

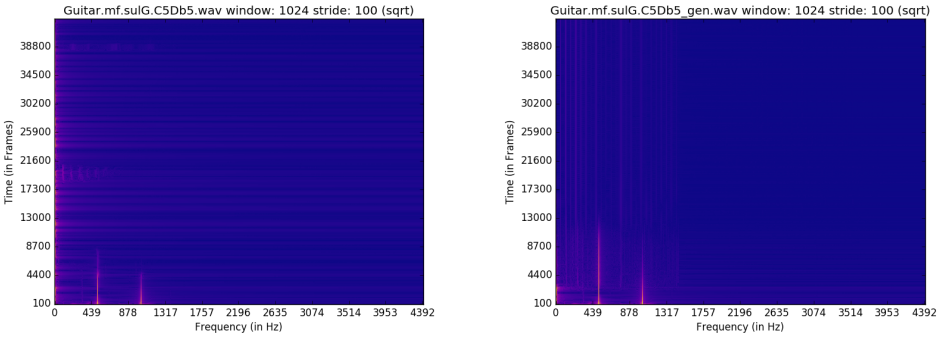

Figure 4: STFT spectrograms of guitar $C5$ note. *Left* the original signal. *Right* the generated signal

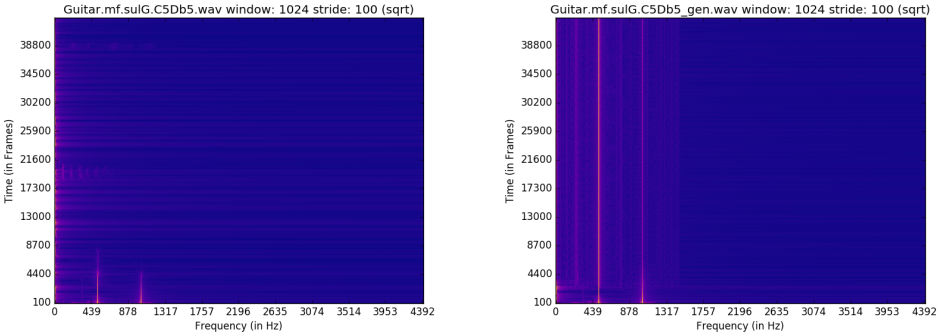

Figure 5: STFT spectrograms of guitar $C5$ note not in training set. *Left* original *Right* generated

Guitar: As mentioned above, the guitar data-set is considerably simpler, as it consists mostly of the note C and has relatively few significant overtones, which also allowed for significant downsampling. It should therefore be no surprise that it is also, where we obtained the best results. In figure 4 we see that, when primed on a signal from the training set, the model generates a signal which decays slower than the original but in a similar fashion and without much added noise. Therefore this is the one data-set, on which we can produce results, which meet each of our qualitative requirements. When the same signal is excluded from the training, however, the model can no longer model the decay and instead simply copies the same signal indefinitely, as shown in figure 5.

Cello: On the cello data-set noise levels are considerably higher and the decay is shorter than in the original samples. as can can be seen in the example in figure 6.

Piano: Much like the cello data, the piano data-set produces high levels of noise and shortened decay. In an attempt to make the data-set more homogeneous, we reduced it to the A-notes across octaves, which does reduce noise, but also leads to overfitting, where the model produced the signal of another pitch alongside the primed one, as can be heard (and barely seen) in the sample associated with figure 7, where around 3.5 kHz an A two octaves higher begins playing after the prime.

Flute: The flute data-set was used early in the project to test, whether the model could learn a static signal, but then abandoned, because flute has no, or rather a near instantaneous, decay.

The second model has has been tested in varying setups, but so far none of them have shown any sign of learning, returning random noise regardless of the prime sequence used. Audio files on all data-sets including the results referenced here are available on our github page[7].

[7] `https://github.com/Fr-d-rik/generative_audio/tree/master/results`

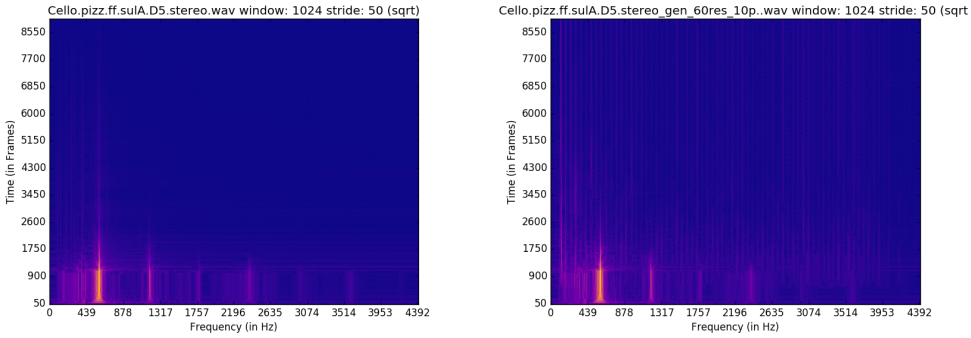

Figure 6: STFT spectrograms of cello (pizzicato) $D5$ note. *Left* the original signal. *Right* the generated signal

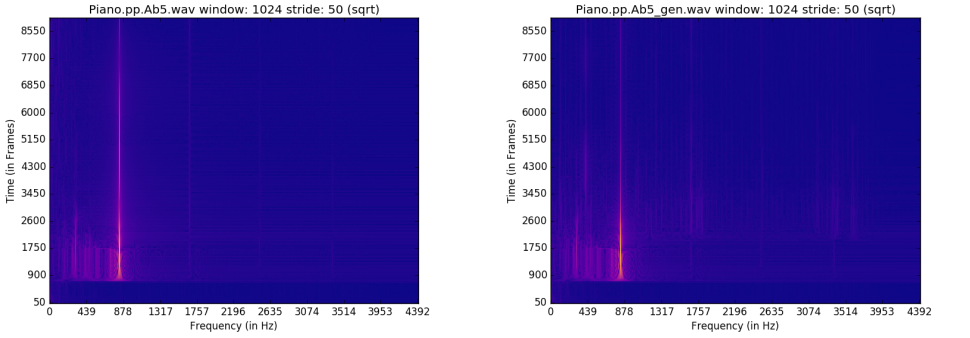

Figure 7: STFT spectrograms of piano $Ab5$ note. *Left* the original signal. *Right* the generated signal

5 Discussion and future work

Overall the results we obtained in this project have been a bit sobering. While we were able to establish a proof of concept with our basic model on the guitar data-set, none of the modifications and additions we have made to the basic setup seem to have a beneficial effect. We have shown that recurrent networks with LSTM units can model simple raw audio signals and model their dynamics over time without adding significant levels of noise. There is still plenty of room for improvement on both the temporal accuracy of the decay and the noise, which although not as present is of course still nonzero, but we believe, that the big next challenge is that of generalizability. Almost all the experiments we present here evaluate on training data, and as such test the system's capacity for overfitting. In the one case where we test on unseen guitar data, the model fails to generate decay, even though other samples of the same pitch are in the training set. A model which can generalize to unseen pitches may well require changes in architecture and the training regime. For further projects, one also needs to re-evaluate, what constitutes a sufficiently large and diverse data-set. Given the time-constraints we had to take what was readily available and because we lack experience working with audio data, we had little intuition on what a data-set for this task should look like. So far training times on a DAS4 node for our models range from around 10 to 30 minutes, which means that a larger data-set would be easily feasible.

As mentioned earlier, the training of the CNN-LSTM model did not reveal any fruitful results so far. We can conclude that the model has not learned anything useful. Due to our inexperience with the Theano framework we were not able to easily reveal the reasons for these results in full detail. We have started analyzing the weights of the trained model for the recurrent and convolutional layers. The trained weights show a normal distribution with approximately zero mean and a standard deviation between 0.03 and 0.05 which seems to be reasonable for these models. The next step would be to get hold of the gradients that are calculated after each minibatch for each layer and time slice. It is

possible that the gradients in the convolutional layer are quickly converging to zero without having extracted any useful localized features of the signal.

It is tempting to investigate the possibilities of more powerful models that were developed in the last three years for modelling high dimensional sequence data like speech, audio and handwriting. We are referring to the papers of Chung et al. [10], Boulanger-Lewandowski [4], Bayer and Osendorfer [6], Fabius and Amersfoort [7] and Gregor et al. [12]. Many of these researchers argue that the way in which an RNN models the output variability, which can be a unimodal or a mixture of unimodal distributions, is an inappropriate way to model the kind of variability observed in highly structured data such as audio or speech, which is characterized by strong and complex dependencies among the output variables at different timesteps. Instead their RNN models are equipped with a layer of latent random variables to model the variability observed in the data. These models mostly combine an RNN with a Restricted Boltzman Machine (RBM) or a Variational Auto Encoder (VAE). Although we think using one of these models for the learning task at hand is worth considering, we have to admit that the single note raw audio signals we use are less complex than the polyphonic music that some of the researchers used in their experiments, and as a result may not be subject to the same problems.

References

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

[2] Eck, Douglas and Schmidhuber, Jürgen. A first look at music composition using lstm recurrent neural networks. Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale, 2002.

[3] Franklin, Judy A. Recurrent neural networks for music computation. INFORMS Journal on Computing, 18(3):321–338, 2006.

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

[5] I-Ting Liu, Bhiksha Ramakrishnan. Bach in 2014: Music Composition with Recurrent Neural Network. arXiv preprint arXiv:1412.3191, 2014

[6] Bayer, Justin and Osendorfer, Christian. Learning stochastic recurrent networks. In NIPS 2014 Workshop on Advances in Variational Inference, 2014.

[7] O. Fabius, J. R. van Amersfoort, and D. P. Kingma. Variational recurrent auto-encoders. arXiv preprint arXiv:1412.6581, 2014.

[8] Cho, Kyunghyun, van Merrienboer, Bart, Gulcehre, Caglar, Bougares, Fethi, Schwenk, Holger, and Bengio, Yoshua. Learning phrase representations using rnn encoderdecoder for statistical machine translation. arXiv preprint arXiv:1406.1078, 2014.

[9] Aran Nayebi and Matt Vitelli. GRUV: Algorithmic Music Generation using Recurrent Neural Networks. `https://cs224d.stanford.edu/reports/NayebiAran.pdf`, 2015

[10] Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron Courville, Yoshua Bengio. Recurrent Latent Variable Model for Sequential Data. arXiv preprint arXiv:1506.02216, 2016

[11] Jakub Fiala, blog: Deep Learning and Sound, `http://fiala.uk/notes/deep-learning-and-sound-01-intro`, 2015

[12] K. Gregor, I. Danihelka, A. Graves, and D. Wierstra. Draw: A recurrent neural network for image generation. In Proceedings of The 32nd International Conference on Machine Learning (ICML), 2015.