# Report Deep Learning Lab

Anna-Lena Popkes, Pascal Wenker

August 9, 2017

**Abstract**

In this paper, we investigate the ability of recurrent neural networks with Long Short-Term memory to learn melody lines, rhythm and local- and global structures in monophonic musical sequences. We test different architectural features and provide experimental results showing that the resulting models are capable of generating music with complex, long-range structure. We further provide example sequences generated by the networks.

## 1 Introduction

In the last years many fields of study showed great interest in Deep Learning and its various techniques. The task of automatic music generation, which has been subject to research for a long time, has recently gained attention through projects like DeepBach [19] and Google Magenta [25] which have proven the effectiveness of Deep Learning models in the musical domain. The emerging popularity of the field is further underlined by a new workshop on "Deep Learning for music" first held in May 2017 [1].
Because musical data is inherently sequential and rich in underlying structure, many proposed architectures were based on recurrent neural networks (RNNs). RNNs are powerful models specialized for processing sequences. They have produced state of the art results in tasks like language modelling, speech recognition [16, 15] or machine translation [3]. Regarding the *generation* of sequences they have previously been used in domains like text [14, 31] or on motion capture data [30]. Because recurrent networks incorporate internal feedback loops and a memory of previous events they are in theory able to model sequences of arbitrary complexity. However, in practice, recurrent models have difficulties with learning long-term dependencies. In musical data, patterns spanning across multiple time steps are especially important. While humans have an intuitive understanding of such long-term structures and can easily identify and understand them, machine learning models like RNNs need to be designed carefully to incorporate both long- and short-term memory.

In this context we wish to exploit the ability of recurrent architectures to generate novel monophonic melodies. In our design we consider both short-term and

long-term memory by using Long Short-Term Memory layers and an attention mechanism. The remainder of the paper is organized as follows: in Section 2 we present previous work on monophonic and polyphonic music generation. Section 3 introduces the basic architecture of recurrent networks and Long Short-Term Memory. Also, we discuss how recurrent models can be used for generating sequences. In Section 4 we present the Google Magenta project which served as an inspiration for our work. In Section 5 we present our model, data and results. Finally, a conclusion is given in Section 6.

## 2 Related Work

The beginnings of music generation with recurrent neural networks were based on using recurrent models as single-step predictors, generating music on a note-by-note basis. Such an approach was first used by Todd [34] and Baruchal and Todd in 1989 [5]. Todd used a Jordan recurrent neural network trained on monophonic melodies with Backpropagation Through Time (BPTT) and teacher forcing. Although the model was able to learn and generate musical lines, the generated sequences were short and lacking "overall global organization" [34, page 42].

Mozer [28] proposed an architecture based on a modified Elman network which produced slightly better results. However, it was still not able to capture global musical structure.

In [9] the network of [34] was adopted and a second training phase was added. The second phase used reinforcement learning to calculate a scalar value describing the quality of the output, which replaced the explicit error information.

To deal with the vanishing gradient problem and to learn long-term dependencies and global structures in music, Eck and Schmidthuber [7] investigated the suitability of Long-Short Term Memory (LSTM) architectures for algorithmic composition. They successfully trained an LSTM architecture on blues music and demonstrated its ability to capture both the local structure of music and the long-term structure of a musical style.

Boulanger-Lewandowski et al. [6] proposed an RNN based architecture using a Restricted Boltzmann machine (RBM) for the task of polyphonic music generation. The basic *recurrent neural network - Restricted Boltzmann machine* (RNN-RBM) presented by the authors was able to learn basic harmony rules, melody lines and local temporal coherence. However, it was not able to capture or generate long-term structures. The authors proposed several extensions to improve their model. For example, they replaced the RBM with a neural autoregressive distribution estimator (NADE) which allowed for exact gradient computation.

Goel et al. [11] modified the model of Boulanger-Lewandowski et al. by replacing the RBM with a Deep Belief network (DBN), forming a *recurrent neural network - Deep Belief network* (RNN-DBN). A DBN consists of a stack of single RBMs trained sequentially. Unlike Restricted Boltzmann machines, Deep Belief networks learn to extract a deep hierarchical representation of the given training data. This enabled the model to produce more complex melodies without significant repetitions. A further modification was made by Vohra et al. [35] who replaced the RNN with an LSTM, improving the capability of the architecture to model temporal dependencies across several time steps. Lyu et al. [23] took a similar approach and combined an LSTM with a recurrent temporal Restricted Boltzmann machine (RTRBM).

More recently, other architectures were proposed. Hadjeres et al. [19] outlined several drawbacks of the approaches taken by [6, 11, 35, 23] and of other agnostic approaches for polyphonic music generation. For example, Hadjeres et al. described the models as being too general to capture specific styles (e.g. the style of Bach) and not flexible enough to allow for user interaction. As a possible solution the authors presented a model based on a dependency network, pseudo-Gibbs sampling and a specific representation of musical data. The proposed model was able to produce four-part chorales in the style of Bach and allowed for user interaction.

Since the start of the Google Magenta project in June 2016 [25] (see Section 4) several architectures based on Deep Learning were proposed. For example, the Google Magenta team recently published a paper introducing a new model together with a large, publicly available dataset [8] in order to encourage other researchers to use the dataset as a benchmark and entry point into the topic of audio machine learning.

# 3 Recurrent Neural Networks

A recurrent neural network (RNN) is a special type of neural network for processing sequential data. It differs from a feedforward neural network by containing recurrent (cyclic) connections. Many different architectures for recurrent neural networks exist. As pointed out by Goodfellow et al. [12, chapter 10]: "Much as almost any function can be considered a feedforward neural network, essentially any function involving recurrence can be considered a recurrent neural network".

Different to feedforward models, recurrent networks share the same weights across multiple time steps. The sharing of parameters arises from the way in which an RNN operates: each member of the output is produced by applying the same update rule to each member of the previous output. When using a basic RNN the values of the hidden units $\boldsymbol{h}$ can be described as follows [12]:

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}, \boldsymbol{\theta}) \tag{1}$$

with $\boldsymbol{x}^{(t)}$ being the input vector at time step $t$ and $\boldsymbol{\theta}$ being the parameters of the model. Furthermore, the model typically contains an output layer that uses information from the hidden state in order to make predictions. Because the state of a neuron at time step $t$ is a function of all inputs from previous time steps, the recurrent connections can be viewed as creating a kind of "memory". A hidden unit that preserves information across multiple time steps is called a *memory cell* or *cell* [18, 12, 14]. In the following, we will describe a basic type of cell for which the transition from one hidden state to the next is based on an affine transformation followed by a point-wise nonlinearity. Several other types of cells exist. One of them, Long Short-Term Memory cells, are discussed in chapter 3.1.

An example of a basic recurrent network is illustrated in figure 1. The figure shows the computational graph of a recurrent neural network that maps an input sequence $\boldsymbol{x}$ to an output sequence $\boldsymbol{o}$. The loss $L$ measures how far each output $\boldsymbol{o}$ is from the corresponding training target $\boldsymbol{y}$. For example, when using a softmax output layer, the outputs $\boldsymbol{o}$ are considered to be unnormalized log probabilities. The loss then internally computes $\hat{\boldsymbol{y}} = \text{softmax}(\boldsymbol{o})$ to receive normalized probabilities and compares these values to the targets $\boldsymbol{y}$. The input to hidden connections of the network are parametrized by a weight matrix $\boldsymbol{U}$, the hidden-to-hidden connections are parametrized by a weight matrix $\boldsymbol{W}$ and the hidden-to-output connections are parametrized by a weight matrix $\boldsymbol{V}$ [12]. For an input vector sequence $\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(\tau)}$ and an initial hidden state $\boldsymbol{h}^{(0)}$ the activations of the hidden units for $t = 1$ to $t = \tau$ are computed as follows [14, 12]:

$$\boldsymbol{h}^{(t)} = \sigma_h(\boldsymbol{b} + \boldsymbol{W} \cdot \boldsymbol{h}^{(t-1)} + \boldsymbol{U} \cdot \boldsymbol{x}^{(t)}) \tag{2}$$

where $\boldsymbol{W}, \boldsymbol{V}$ denote weight matrices, $\boldsymbol{b}$ denotes the bias and $\sigma_h$ the activation function in the hidden layer. Given the values of the hidden units, the output is computed using the following equation:

$$\hat{\boldsymbol{y}}^{(t)} = \sigma_y(\boldsymbol{c} + \boldsymbol{V} \cdot \boldsymbol{h}^{(t)}) \tag{3}$$

where $\boldsymbol{V}$ denotes the weight matrix, $c$ the bias and $\sigma_y$ the activation function of the output layer (e.g. softmax).
The loss for a sequence of $\boldsymbol{x}$ values and corresponding targets $\boldsymbol{y}$ is given by the sum of the individual losses. If, for example, the loss is given by the negative log-likelihood of $\boldsymbol{y}^{(t)}$ given $\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(t)}$ we have [12]:

$$L(\boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(\tau)}, \boldsymbol{y}^{(1)}, ..., \boldsymbol{y}^{(\tau)}) = -\sum_t \log p_{model}(\boldsymbol{y}^{(t)} | \boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(\tau)}) \tag{4}$$

The partial derivatives of the loss with respect to the parameters can be computed efficiently by applying Backpropagation Through Time to the unrolled

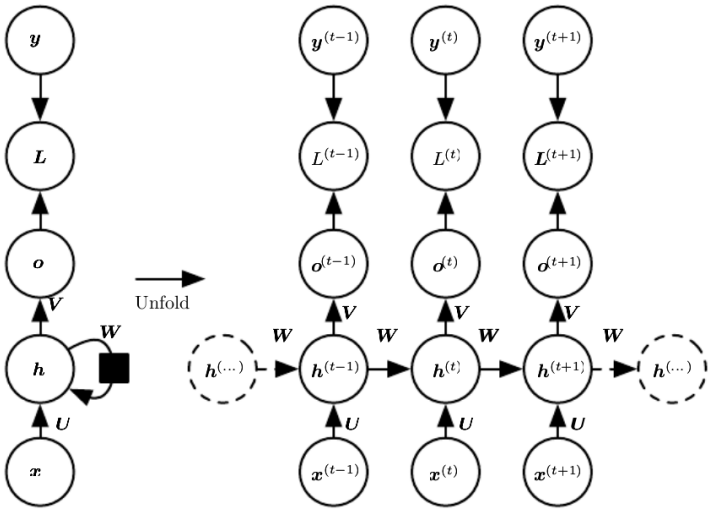


*Figure 1: Computational graph of a recurrent neural network (RNN) that maps an input sequence $\boldsymbol{x}$ to an output sequence $\boldsymbol{o}$. The loss L measures how far each output $\boldsymbol{o}$ is from the corresponding training target $\boldsymbol{y}$. (Left) RNN and loss with recurrent connections. (Right) RNN and loss as an time-unfolded computational graph.*

computational graph (see figure 1) [36]. Afterwards, the network can be trained with gradient descent. Recurrent neural networks are universal approximators to the effect that a recurrent network of finite size can compute any function a Turing machine can compute [12].

One major challenge for deep recurrent neural networks is to learn long-term dependencies. In simple terms, the problem is that gradients propagated over many stages tend to either explode or vanish [20, 4]. Various approaches to solve this difficulty exist. For example, so called 'skip connections' can be added to the network which directly connect variables from the distant past to variables in the present [22]. An approach that has been proven to be extremely effective in the past [14, 38, 37] is to use gated RNNs, such as the Long Short-Term Memory discussed next.

## 3.1 Long Short-Term Memory

The *Long Short-Term Memory* (LSTM) architecture [21] is designed to be better at storing information and at finding and learning long-term dependencies than standard recurrent networks [12, 29, 14]. LSTMs have been extremely successful in a variety of tasks like handwriting recognition [14, 17], speech recognition [16, 15] or machine translation [32].
In many simple RNNs, the activation function of the hidden layer (see equation 2) is given by an element-wise application of some sigmoid function. Gated re-

current networks like LSTMs make use of *gated* activation functions. Long Short-Term Memory architectures contain special *memory cells* which have an internal recurrence (i.e. a self-feedback loop) and additional connections to different gating units that control the flow of information through the cell [29, 14, 12]. The first gate is called the *input gate*. It controls the flow of inputs into the cell and decides whether or not the current input is worth preserving. Similar in function is the *forget gate* which decides how useful the internal state of the memory cell is before feeding it back into the cell through its self-feedback loop. Thereby, it adaptively forgets and resets the cell's memory. The third and last gate is the *output gate*. It controls which parts of the memory cell state should be present in the hidden state, thereby controlling the output flow into the rest of the network [12, 29]. Additionally, the LSTM architecture can contain *peephole connections* from the memory cell to the multiplicative gates [10]. The structure of a single LSTM cell is illustrated in figure 2.

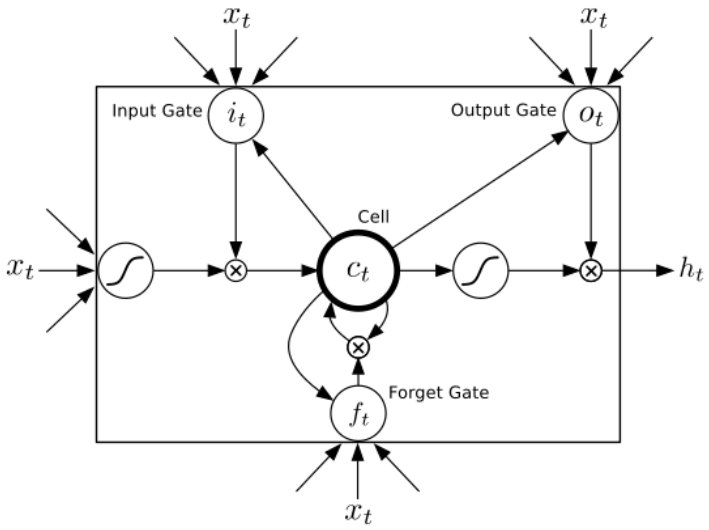


*Figure 2: Structure of a Long Short-Term Memory Cell [14]*

The LSTM used in our model [29] is defined by the following composite function:

$$\boldsymbol{i}^{(t)} = \sigma\big(W_{xi} \cdot \boldsymbol{x}^{(t)} + W_{hi} \cdot \boldsymbol{h}^{(t-1)} + W_{ci} \cdot \boldsymbol{c}^{(t-1)} + b_i\big) \tag{5a}$$

$$\boldsymbol{f}^{(t)} = \sigma\big(W_{xf} \cdot \boldsymbol{x}^{(t)} + W_{hf} \cdot \boldsymbol{h}^{(t-1)} + W_{cf} \cdot \boldsymbol{c}^{(t-1)} + b_f\big) \tag{5b}$$

$$\boldsymbol{c}^{(t)} = \boldsymbol{f}^{(t)} \cdot \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \cdot \tanh\big(W_{xc} \cdot \boldsymbol{x}^{(t)} + W_{hc} \cdot \boldsymbol{h}^{(t-1)} + b_c\big) \tag{5c}$$

$$\boldsymbol{o}^{(t)} = \sigma\big(W_{xo} \cdot \boldsymbol{x}^{(t)} + W_{ho} \cdot \boldsymbol{h}^{(t-1)} + W_{co} \cdot \boldsymbol{c}^{(t)} + b_o\big) \tag{5d}$$

$$\boldsymbol{h}^{(t)} = \boldsymbol{o}^{(t)} \cdot \tanh(\boldsymbol{c}^{(t)}) \tag{5e}$$

where $\boldsymbol{i}$, $\boldsymbol{f}$, $\boldsymbol{c}$ and $\boldsymbol{o}$ are respectively the *input gate, forget gate, (memory) cell* activation vectors and the *output gate*, all of which have the same size as the hidden vector $\boldsymbol{h}$. $\sigma$ denotes the logistic sigmoid function, and each $W$ term denotes a weight matrix (e.g. $W_{hi}$ is the weight matrix between the input gate and the hidden gate). $W_{ci}, W_{cf}, W_{co}$ are diagonal weight matrices for peephole connections and $b_i, b_f, b_o$ are respectively the input gate, forget gate and output gate bias vectors.

### 3.2 Sequence Generation

As outlined in Section 2 the simplest way to generate sequences with a recurrent neural network is to use it as a single-step predictor. During training, the network is given the event at time step $t$ as an input and the event at time step $t + 1$ as a target, and learns to predict the value of the next time step given the current input. The predictions are given by probabilities, resulting from applying the softmax function to the unnormalized log-probabilities computed by the network. After training, a new sequence can be generated as follows: we first seed the network with an initial input (e.g. a primer melody or a single note) and sample a value from its output distribution. The sampled value is fed back into the network as a subsequent input. This procedure can be repeated for any number of steps, generating arbitrary long sequences [18, 7, 14].

As discussed before (see Section 3) RNNs have problems with learning long-term dependencies. This characteristic makes it difficult for recurrent networks to recover from past mistakes when being applied to the task of sequence generation. If the predictions made by the network are based on only a few of the previous inputs, and if these inputs were themselves generated by the network, past mistakes will likely affect the entire generation process. To prevent a network from showing such an instability we can adjust its cells to having a larger memory, for example by using LSTMs. In this way, the network can use information from the distant past if the recent history is inconsistent [14]. We can further add an attention mechanism, allowing the network to consider not only the last, but the last $n$ time steps. This is further described in Section 4.1.

## 4 Magenta

Google Magenta is a project from the Google Brain team that tries to use machine learning for creating art and music [25]. Since the project was launched on 1st June, 2016 the Magenta group has continuously been adding new software on its GitHub page and released regular blog posts on its work. The Magenta project was inspired by DeepDream, a visualization tool designed to understand both the functioning of neural networks and the features learned by its individual layers [13, 2]. The results of DeepDream suggested that neural networks

"could be used for artistic purposes" [2].

The Magenta project has two major goals. In the past years a lot of progress has been made in tasks based on *understanding* content, like speech recognition [16, 15] or translation [37, 3]. The first goal of Magenta is to advance the state of the art in tasks concerned with *generating* content in music and art. This makes Magenta one of the biggest research projects in this field.
Magenta's second goal is to build a community of machine learning researchers, programmers and artists that develop, discuss and employ the tools and models in a collaborative fashion [25]. All code developed by the core Magenta team is published on their GitHub page and extended by code from external contributers. Furthermore, a discussion group encourages the exchange of ideas and problems [24].
In addition, Magenta outlines four main research goals. The first and prior goal of the project is to design algorithms capable of generating art and music. The second goal is to incorporate effects like attention and surprise in the machine-learning models. Thirdly, and named as the biggest challenge is the task of storytelling. The fourth research goal is concerned with the evaluation of designed models. In order to compare developed models Magenta wants to include artists and musicians in their projects and use their expertise to evaluate the models and tools.

At the time of this paper the Magenta project already compromises nine different models. Our work was inspired by the "Melody RNN" model. Like all Magenta code the Melody RNN uses TensorFlow, Google's open-source library for machine learning [33].

## 4.1 Melody RNN

The Melody RNN was the first model released by the Magenta team [27]. It applies the ideas of language modelling to the task of melody generation. The model is based on a recurrent neural network with Long Short-Term Memory (LSTM) cells which is is able to predict musical notes given a sequence of previous notes. Three different configurations of the model exist. The basic configuration acts as a baseline for the task of melody generation. It uses a one-hot encoding of the previous event as an input to the network. The label is given by the one-hot representation of the next event in the sequence. Possible events are: note-off (indicating to turn off any notes currently playing), a different note-on event for each pitch and no event (indicating to either keep playing a note or continuing silence). The other two configurations, Lookback and Attention, are focused on improving the ability of the network to recognize and reproduce long(er)-term structures.
The Lookback configuration includes additional information in the input vector and labels. First, not only the previous, but also events from one or two bars ago are fed into the network. This information allows the network to recognize short patterns (e.g. mirrored melodies). Second, the input includes whether

the last event repeated the event from one or two bars before it, allowing the network to recognize repetitive patterns. The last additional input information is given by a binary representation of the current position within the measure, simplifying learning patterns associated with 4/4 time music. The labels are not further given by the next event in the sequence. Instead, labels are either "repeat-1-bar-ago" or "repeat-2-bars-ago". Only if the melody is not repeating, the label is given by the next event. The modified labels make it easier for the model to repeat patterns that occur across one or two bars because it does not have to store the sequences in its memory cell [26].
The Attention configuration incorporates a mechanism similar to the one used in our model. We base our attention configuration on the attention method introduced in a paper by Bahdanau et al. [3]. When using attention, a recurrent network does not only consider the last time step when generating the output for the current time step. Instead, we allow the network to "attend" to the last $n$ time steps. During the training procedure the network learns which aspects to attend to based on the input sequence and the previous outputs. Therefore, the output of the network does not depend only on the last state anymore but on a *weighted combination* of the $n$ previous states. This allows the model to access information from the past without having to store that information in the state of the cell. The effects of attention are further discussed in Section 5.3.

In addition to the three models, the Melody RNN provides code to extract monophonic melodies from a set of MIDI files and transform them into the native TensorFlow format.

# 5 Model

## 5.1 Training Data

We evaluate and compare the performance of different LSTM RNN architectures on two MIDI datasets:

**Weimar Jazz database**: small dataset of 456 jazz solo transcriptions
**Lakh MIDI dataset** (clean midi version): a subset of MIDIs from the original Lakh dataset which contains 176581 unique MIDI files

We use open-source code provided by Magenta to extract monophonic melodies from the MIDI files of all datasets. We use the baseline version described in Section 4.1. To facilitate analysis and comparison between architectures, all extracted sequences are transposed in a common tonality (C major) and the tempo is normalized to 120 quarter notes per minute. During training, the sequences are padded with zeros to the length of the longest sequence in each input batch. Finally, the melodies generated by the network are converted into MIDI files using Magenta.

## 5.2 Model architecture

We use a recurrent neural network with two LSTM layers, each having 128 LSTM units. The corresponding equations are given in Section 3.1. The input and output layers have 38 units, corresponding to the size of the input. We use a softmax output layer. For each dataset we compare the effects of using a dropout probability of 0.5, refeeding the final LSTM state during training, and of including the attention mechanism described in Section 4.1.
All parameter gradients are computed using truncated backpropagation through time (see Section 3). The Adam algorithm is used to optimize the network parameters. All models are trained with a learning rate of 0.001. In case of the Jazz dataset each model is trained until convergence (6900 training steps), in case of the Lakh dataset the models are trained for 55800 training steps due to limited computational resources. To prevent the gradients from exploding, we apply gradient clipping using an $L2$ norm of 5.
We evaluate the architectures quantitatively by using their loss and accuracy as performance measures and qualitatively by generating sample sequences.[1] As a baseline we use the untrained model to generate sample sequences.

## 5.3 Results

In tables 1 and 2 we summarize the results for various LSTM RNN architectures. We make several key observations:

- When refeeding the final LSTM state during training the quality of generated sequences is improved substantially for small datasets like the Jazz solos (see table 1 models j1, j2). We believe that the differences in performance can be attributed to the nature of the datasets. In case of the Jazz dataset, sequences in successive batches are related to each other in a logical way, belonging to the same category of music and often being composed by the same artist.

- The perceived quality of generated sequences is improved when applying dropout, even when training on a small dataset. This is supported by the results computed on a separate test set (see tables 1, 2). As evident in the tables, the test results are significantly better for the models using dropout. This suggests that overfitting has a negative effect on a model's ability to capture musical lines, short- and long-term structures in musical data.

- When not refeeding the final state (Jazz dataset), additional characteristics like dropout and attention have only little effect on the perceived quality of the produced sequences. In this setting, contrary to our expectations, the model using only dropout (j4) performed better during sequence generation than the model using both dropout and attention

[1] Samples of generated melodies and training inputs can be downloaded at `https://github.com/zotroneneis/deep-music/`

(j5). Using dropout does not prevent the model from overfitting. This becomes evident in the low training loss (0.11) and high training accuracy (0.99) of model j5. However, overfitting results in a weak performance on the test set and during sequence generation, as evident in table 1.

- In case of the Lakh dataset, sequences in successive batches are not related to each other. Therefore, refeeding the state during training does not improve generated sequences.
- When training the model on a large dataset like Lakh, using attention slightly improves the test accuracy and loss compared to a model using only dropout. The generated sequences sound more varied but the differences are only marginal. This might change when training the models for several epochs. Furthermore, to reduce computational complexity, we chose a small attention window of only twenty steps. A larger window might yield further improvements.
- The generated sample sequences indicate that models using dropout (j1, j4, l1 and l2) have learned local and global structures and are able to create melodic and coherent music. A clear difference to the baseline version can be identified. The generated sequences are improved when training on datasets that belong to one musical category, containing sequences that are related to each other in a logical way.

| identifier | state refeed | dropout | attention | accuracy | loss |
|---|---|---|---|---|---|
| j1 | yes | yes | no | 0.75 | 1.49 |
| j2 | yes | no | no | 0.68 | 3.12 |
| j3 | no | no | no | 0.65 | 3.3 |
| j4 | no | yes | no | 0.71 | 1.85 |
| j5 | no | yes | yes | 0.74 | 5.53 |

*Table 1: Results on Jazz test dataset*

| identifier | state refeed | dropout | attention | accuracy | loss |
|---|---|---|---|---|---|
| l1 | no | yes | no | 0.778 | 0.86 |
| l2 | no | yes | yes | 0.787 | 0.76 |

*Table 2: Results on Lakh test dataset*

# 6 Conclusion and future work

This report presented different recurrent architectures based on Long Short-Term memory and their ability to learn melody lines, rhythm and local- and

global musical structures. We were able to show that LSTM networks are capable of generating new melodic sequences with short- and long-term structure. Further, we showed that refeeding the final state of the network during training can improve its performance for datasets with related sequences.
Several directions for future work suggest themselves. First, as mentioned before, we used open-source code by Magenta to convert our MIDI files to the standard TensorFlow format. To further improve the network's ability to recognize and reproduce long-term structures all experiments could be repeated using the input representation of Magenta's lookback configuration (see Section 4.1).
Unfortunately, we were not able to successfully train a model using both attention and refeeding the final LSTM state during training. Although our results suggest that attention increases the amount of overfitting in the case of small datasets, this has to be supported by additional experiments. Furthermore, we restricted the project to comparing different architectural characteristics. Varying the number and sizes of the LSTM layers could yield further improvements.
To prevent numerical instabilities during training, caused by the large number of training steps, we had to adapt the epsilon parameter of the Adam optimizer from 1e-08 to 1e-01 when training on the Lakh dataset. Future experiments should employ an epsilon value of 1e-01 from the beginning, preempting any problems of this kind.
It would also be interesting to evaluate the effect of adapting the training procedure. In all experiments, the networks were trained with teacher forced inputs. Hence, the inputs encountered during generation likely varied from the input conditions seen during training. As outlined in [12] these effects can be mitigated by training the models with both free-running and teacher-forced inputs.
Lastly, we evaluated our models using their accuracy and loss. However, these metrics do not necessarily correspond to the human perception of musical quality. A high accuracy or low cost do not guarantee that the generated melodies are perceived as pleasing. Further studies are needed to design better evaluation metrics.

# References


[1] International workshop on deep learning for music, May 2017. `http://dorienherremans.com/dlm2017/`.

[2] Research at Google. Music and art generation. `https://research.google.com/teams/brain/music-and-art/`.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[4] Yoshua Bengio, Paolo Frasconi, and Patrice Simard. The problem of learn-

ing long-term dependencies in recurrent networks. In *Neural Networks, 1993., IEEE International Conference on*, pages 1183–1188. IEEE, 1993.

[5] Jamshed J Bharucha and Peter M Todd. Modeling the perception of tonal structure with neural nets. *Computer Music Journal*, 13(4):44–53, 1989.

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

[7] Douglas Eck and Juergen Schmidhuber. A first look at music composition using lstm recurrent neural networks. *Istituto Dalle Molle Di Studi Sull Intelligenza Artificiale*, 103, 2002.

[8] Jesse Engel, Cinjon Resnick, Adam Roberts, Sander Dieleman, Douglas Eck, Karen Simonyan, and Mohammad Norouzi. Neural audio synthesis of musical notes with wavenet autoencoders. *arXiv preprint arXiv:1704.01279*, April 2017.

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

[10] Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber. Learning precise timing with lstm recurrent networks. *Journal of machine learning research*, 3(Aug):115–143, 2002.

[11] Kratarth Goel, Raunaq Vohra, and JK Sahoo. Polyphonic music generation by modeling temporal dependencies using a rnn-dbn. In *International Conference on Artificial Neural Networks*, pages 217–224. Springer, 2014.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[13] Google. Inceptionism: Going deeper into neural networks, June 2015. `https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html`.

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

[15] Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 1764–1772, 2014.

[16] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

[17] Alex Graves and Jürgen Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. In *Advances in neural information processing systems*, pages 545–552, 2009.

[18] Aurlien Gron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. O'Reilly Media, March 2017.

[19] Gaëtan Hadjeres and François Pachet. Deepbach: a steerable model for bach chorales generation. *arXiv preprint arXiv:1612.01010*, 2016.

[20] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91, 1991.

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

[22] Tsungnan Lin, Bill G Horne, Peter Tino, and C Lee Giles. Learning long-term dependencies in narx recurrent neural networks. *IEEE Transactions on Neural Networks*, 7(6):1329–1338, 1996.

[23] Qi Lyu, Zhiyong Wu, Jun Zhu, and Helen Meng. Modelling high-dimensional sequences with lstm-rtrbm: Application to polyphonic music generation. In *IJCAI*, pages 4138–4139, 2015.

[24] Google Magenta. Magenta discussion group. `https://groups.google.com/a/tensorflow.org/forum/`!forum/magenta-discuss.

[25] Google Magenta. Welcome to magenta! `https://magenta.tensorflow.org/welcome-to-magenta`.

[26] Google Magenta. Generating long-term structure in songs and stories, July 2016. `https://magenta.tensorflow.org/2016/07/15/lookback-rnn-attention-rnn/`.

[27] Google Magenta. A recurrent neural network music generation tutorial, June 2016. `https://magenta.tensorflow.org/2016/06/10/recurrent-neural-network-generation-tutorial`.

[28] 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.

[29] Haşim Sak, Andrew Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[30] Ilya Sutskever, Geoffrey E Hinton, and Graham W Taylor. The recurrent temporal restricted boltzmann machine. In *Advances in Neural Information Processing Systems*, pages 1601–1608, 2009.

[31] Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024, 2011.

[32] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[33] Tensorflow. An open-source software library for machine intelligence. `https://www.tensorflow.org/`.

[34] Peter M Todd. A connectionist approach to algorithmic composition. *Computer Music Journal*, 13(4):27–43, 1989.

[35] Raunaq Vohra, Kratarth Goel, and JK Sahoo. Modeling temporal dependencies in data using a dbn-lstm. In *Data Science and Advanced Analytics (DSAA), 2015. 36678 2015. IEEE International Conference on*, pages 1–4. IEEE, 2015.

[36] Ronald J Williams and David Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. *Backpropagation: Theory, architectures, and applications*, 1:433–486, 1995.

[37] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[38] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning*, pages 2048–2057, 2015.