LSTM Networks for Music Generation

Eliza Cohn
ecohn4@jhu.edu

Abstract

This paper presents a computationally efficient method for generating artificial music samples. This was accomplished through the use of an LSTM network. This network architecture was selected for use due to their built in memory units, making them an ideal candidate for learning long-range dependencies over a data set. Selecting a database of Irish folk songs as the training set, each song was numerically encoded to be the input. By feeding the output of the LSTM network into a fully connected layer, the softmax could be taken over this final layer. This created a probability distribution that could be optimized to produce the highest quality generated samples through training.

1. Introduction

Since the inception of computers and the conception of artificial intelligence, melodic composition has long been thought of as the ultimate test of computational complexity [12]. The generation of music is now a very attainable goal, due to the development of neural networks and various numerical representations of music.

The use of neural networks allows for the implementation of probabilistic models to determine the generated note at each time step. Through the structure of neural networks, the training of these models is already widely understood. It is achieved through initialization of the network parameters and tuning them using gradient based optimization. The loss function used (cross entropy, mean squared error, etc.) determine the way the training loss is defined and influence the way in which the parameters achieve their final values [13].

The key to successful training begins with the way the music is represented and fed as an input into the network. In order to get a meaningful output, the model must map not only the pure notes of the sample onto a probabilistic model, but also the measures, tempo, and keys [11]. Therefore, it is of critical importance that this information is preserved into the structure of the encoded music, for the model is learning the combination of all of the above patterns. It is important to note that the more complicated the structure of the music, the more complicated the network architecture must be. For this reason, the most common type of music generation is monodic, meaning a single melody is generated from the network. More complicated structures are polyphonic, meaning both a harmonic and melodic voice are present in the samples [3]. Therefore, if one wishes to produce polyphonic output, the input must be polyphonic as well for the network to be able to suitably learn the relations. Once the model is trained, one can sample from the final layer of the network to create the generated sample.

The type of network used is of supreme importance to successful generation. The classical feed-forward network is of little use in the composition of music. This is because it has no ability to store any information besides from the previous pass through. When trying to create something with an overall structure, such as a song, this will prove to be a hindrance. This is why recurrent neural networks are the prime selection to be the basis for music generation architectures. By introducing memory units, the previous information of each state is preserved and fed into the input of the next sEq uential state. In this way, the output of each time step is conditioned on the input information of the previous time step [17].

Long Short Term Memory Units (LSTM) are a variation of RNNs. They now include update gates to decide whether to pass the previous state information to the next cell. In addition, to update gates, LSTMs include forget gates which allow further control over which information is preserved in the state information. Finally, the output gate controls which values will be moving forward to the next hidden state. Mathematically, LSTM networks are of importance because they solve the problem of the vanishing gradient, where the input gradients tend towards zero over training. This prevents the learning of long range dependencies over time, and is a common problem of most RNNs.

Therefore, to create music, several key components have been identified. The architecture must include an encoding process, which takes the raw music files to be used as input and creates a numerical representation of them. In addition, some type of architecture must be used that can create a probabilistic model of the collection of samples that can be later sampled to generate the music. LSTMs are widely used, though other methodologies will be addressed in the following section. Finally, a metric to determine the loss function between the training samples and the target samples must be selected in order for the back-propagation update of the network parameters to be achieved. This combination of steps will allow the final output to be decoded and reconstructed into a familiar musical format to be played back by the listener.

2. Related Work

Music Generation has been a widely studied topic that has evolved in the past decades[9]. Before the advent of neural networks, Markov Models were one of the original techniques used as the basis for a music generation system. In 1956, Pinkerton published his original study focusing on information theory and melody [15]. Because Markov Models describe a sEq uence of possible events, they are well suited to make predictions on a sEq uence of musical notes. Their limitations lie in the Markov assumption - that the probability of the next state depends solely on the previous state. This prevents long range dependencies from forming in the final output.

Generative Adversarial Networks (GANs) are also a prevalent method in generating music. The core of the model is the construction of two networks - the generator and the discriminator. The generator trains a probability model to be sampled as a believable output. The discriminator seeks to distinguishes between the "fake" samples and the input data. The training procedure is the minimax game between the generator and discriminator [6]. In this way, the networks are trained to generate samples that mimic reality as closely as possible, making them well-suited candidates for generating synthetic music.

Google Magenta [8] is one of the most popularized and well-respected opensource implementations of the GAN technique for music generation. Magenta is an open source research project that investigates the role of machine learning in the creative process, giving users the tools to experiment with their own music generation without having to write the code themselves. Projects like Magenta Studio and NSynth are hallmark tools in creating new sounds. MuseNet is another example of a neural network developed for users to generate and experiment with musical composition in a user-friendly way [14]. It uses a diverse database of MIDI files as input to their sparse transformer, which is essentially a deep neural network that predicts sEq uences with context-dependent sparsity restrictions.

There are many ways to represent musical files. MIDI files are just one way to capture and compress the contents of an audio file. Feature extraction can also be used in the pre-processing stage by taking the short term Fourier transform (STFT) of the sample. Two types of features can be extracted - short-term and long-term. Short-term features which are extracted over a window of the sample, while long-term features which are extracted from a combination of short term features [2]. Some examples include spectral centroid, spectral rolloff, and Mel-frEq uency Cepstral Coefficients. These features condense the information present in the sample to be as non-redundant as possible, while highlighting different aspects of the music not obvious to the human listener.

The loss function implemented is also a source of much study. In this paper, the classical loss measure of cross entropy was used to compute the loss during training. However, there are several other techniques that can be used to compare musical samples. The inception score is a metric first popularized in use of GANs, proposed by Salimans et al. [16]. It is a measure of both the diversity and semantic discriminability in the generated sample. It has been found that this measure is very similar to human judgment in the quality of a generated sample. The Frechet Inception Distance (FID) was proposed by Heusel et al. as an improvement to the inception score in 2017 [10]. The FID score not only estimates the quality of a collection of generated samples, but also captures how close the synthetic samples compare to real samples, which is not done by the inception score.

3. Methodology

There are multiple steps that must be implemented on order to create a music generation system. First, an appropriate dataset was selected. Then, the pro-processing algorithm was selected to turn the raw music file into a format suitable as an input for a neural network. Then, the architecture was defined, in determining the order and amount of LSTM layers and the output layer. The loss function was then selected and encoded into the system. Finally, the generated output was generated from the resulting probabilistic model.

3.1. The Dataset

The data set selected consisted of 816 classical Irish folk tunes, compiled by MIT 6.S191 [1]. Each audio file is monodic, meaning a single instrument is playing one melody throughout the entire duration of the clip. The clips are all approximately 60 seconds long and originally in a MIDI file format. It is noted that the data set is not particularly large or diverse - as each song is identical in the fact it is represented by a a sole piano voice. This is an important note to make when considering what one will expect the generated output to sound like. Suggestions for methods in diversifying the data set are given at the end of the paper.

3.2. Pre-Processing

The pre-processing method presented in this paper aims to create a numerical encoding of each input song that can be used as input to the neural network. The first process in this is determining the ABC notation of each of the songs. ABC notation the most common form of musical shorthand, using the letters A through G to represent each the pitch of each note present. Additional information of each note is also included such as the sharp, flat, key, and length of the note [19]. There are many such ABC converters available today as open source software, and EasyABC was selected to be used as the converter from MIDI to ABC notation from

Sample Song
Title: A Buachaill Dreoite
Key: G Major

Data:
GF|DGGB d2GB|d2GF Gc (3AGF|DGGB
d2GB|dBcA F2GF|!
DGGB d2GF|DGGF G2Ge|fgaf gbag|fdcA G2:|!
GA|B2BG c2cA|d2GF G2GA|B2BG c2cA|d2DE
F2GA|!
B2BG c2cA|d^cde f2 (3def|g2gf gbag|fdcA
G2:|!

Figure 1. Example of a transcriped song from the data set of irish folk tunes. It is noted that one can see not only the names of the notes encoded, but additional characters representing other sources of information such as key and tempo

the database of software [18]. In total, 83 unique characters are used to encode all of the information provided by the samples. An example of the ABC representation of a sample song is provided in Figure 1.

Once the samples have been processed into ABC notation, it is time to vectorize the samples. Neural networks preform best when operating on numerical inputs as opposed to mixed samples with some characters included. To create the nummerical representation of each sample, two look-up tables were created. One maps the characters to numbers for pre-processing, and the second maps numbers to characters for post-processing.

The final step of pre-processing is the creation of a target sEq uence for each input sample. This is important for calculating the loss during the back propagation portion of training. To create the target sEq uence for each sample song, the text of each sample is broken up to subsEq uences. The target of each subsEq uence is then defined as the input sEq uence but shifted one character over. This allows the network to learn how to predict the following note based on a sEq uence of previous note inputs. This brings to light a hyper-parameter that will need be tuned during training, which the subsEq uence length used in this pre-processing.

To provide an example of target creation, consider the short song "GFDGGB d2GB", which is represented here in ABC notation. Using a subsEq uence length of 5, we see that the first subsEq uence would be "GFDGG", and its target is "FDGGB".

3.3. RNN and LSTM Architecture

Mathematically, RNNs are simply a complicated variation of the traditional feedforward networks. A block diagram has been provided in 2 to help create an intuition of the math provided below.

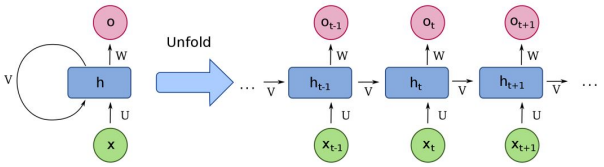

Figure 2. A block diagram representing the folded and unfolded representations recurrent neural network

The principle of RNNs lies in the statement that the current state is a function of the previous state given in Eq 1 and a function of all previous inputs given in Eq 2.

$$h_t = f_\theta\left(h_{t-1}\right), \tag{1}$$

$$h_t = g_t\left(x_t, x_{t-1}, ..., x_1\right), \tag{2}$$

Formally, at any state at time t, the intermediary output a_t is defined as:

$$a_t = b + V * h_{t-1} + u_{x_t}, \tag{3}$$

Where b is the bias term, v are the weights, h_{t-1} is the previous state info, and u_{x_t} is the processed input. Typically, the hyperbolic tan function is used as the non-linearity to compute the final state representation at time t.

$$h_t = \tanh(a_t), \tag{4}$$

The output at time t is computed by passing the state information through another set of learned weights w and bias term c:

$$o_t = c + W * h_t, \tag{5}$$

To create the final probability distribution, the output vector at time t is passed through a softmax layer, similar to the final layer in CNNs. In our case, the softmax layer has an output vector of size 83, to represent each potential character in our ABC notation. Sampling from this distribution produces the note most likely to occur at time t given the prior information.

$$p_t = softmax\left(o_t\right). \tag{6}$$

However, the primary reason for using LSTMs over RNNs in this paper is to solve the problem of the vanishing gradient so often found in RNNs. LSTM networks are very similar to RNNs, with some additional memory and input units [7]. A block diagram of the unfolded and folded representation is provided in Figure 3 to provide intuition to accompany the mathematics.

Again, u_{x_t} represents the processed input at time t and h_t represents the current cell state. We now have the addition of c_t, which represents the current cell memory. There

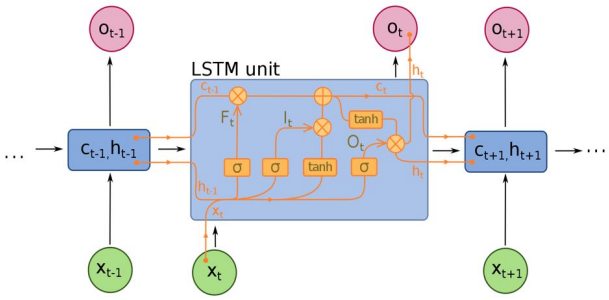

Figure 3. A block diagram representing the folded and unfolded representations of an LSTM

are now multiple weight vectors to correspond to each of the LSTM subunits. W_f, W_i, W_c and W_o correspond to the weights of the forget, cell memory, input, and output cell gates. We can now define the forget gate as

$$f_t = \sigma\left(u_{x_t} + h_{t-1} * W_f\right), \tag{7}$$

Defining the intermediary cell memory gate as:

$$\bar{c}_t = \tanh\left(u_{x_t} + h_{t-1} * W_c\right), \tag{8}$$

Defining the input gate as:

$$l_t = \sigma\left(u_{x_t} + h_{t-1} * W_i\right), \tag{9}$$

Defining the output gas as:

$$o_t = \sigma\left(u_{x_t} + h_{t-1} * W_o\right), \tag{10}$$

Then the resulting cell memory and output can be defined as the following:

$$c_t = f_t * c_{t-1} + l_t * \bar{c}_t \tag{11}$$

$$h_t = o_t * \tanh\left(c_t\right) \tag{12}$$

As before, we can pass this output through a softmax layer to create the final probability distribution to be sampled from, exactly as in Eq 6

3.4. Implementation

The finalized architecture that was implemented is described here. Each time step is represented by three layers. The first layer is the embedding layer, which takes the raw input and encodes it into its numerical representation. That process provides the input for the LSTM layer. The output of the layer is then fed into a fully connected dense layer. The soft-max is taken over this last layer to produce the final output. A block diagram of the network is given in Figure4.

After training the network, the model is now ready to generate output samples to be listened to.

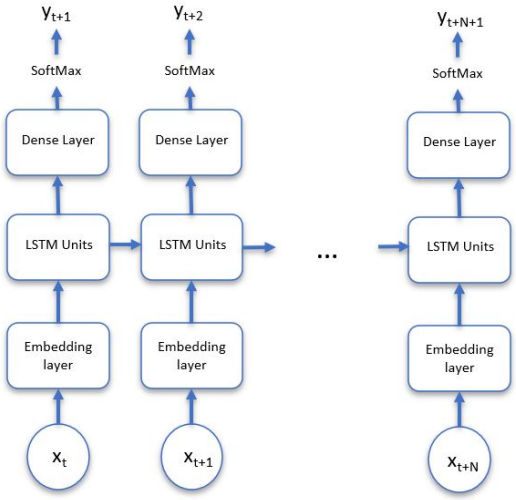

Figure 4. A block diagram representing the final network used.

4. Results

With the network architecture finalized, it could be implemented in TensorFlow using the Keras API [4]. Cross entropy was the loss function used in back propagation for this paper and also a measure of success to allow for the fine-tuning of the network. There are many parameters to optimize over to ensure that the best quality sample is being produced. By comparing which parameter value resulted in the lowest reconstruction error during training allowed for a quantitative analysis to determine the best possible model.

The optimizer used in error propagation was the first to be selected through error evaluation. By testing the most suitable optimizers in training the network, it was determined that the Adam optimizer resulted in the least amount of loss. This can be seen numerically in Figure 5. The Adam algorithm is variation of stochastic gradient descent using an adaptive estimate of second moments. Next, the dimensionality of the LSTM output was determined through error analysis. It is shown quantitatively in Figure 6 the optimal input for the final dense layer with respect to error.

Using the same methods, the learning rate (Figure 7), sequence length (Figure 8) for input dimensionality, and the training batch size (Figure 9) were all determined with respect to the least amount of error. The summary of the final optimal parameters are given in Table 1.

Using these optimal parameters, the best quality music samples could be generated. I have attached the entirety of the network code, along with a sample from the untrained and trained network into my GitHub so that the reader can listen for themselves the difference in quality that a trained network provides [5]. The samples were all saved as .wav files to allow the listener easy access to listening.

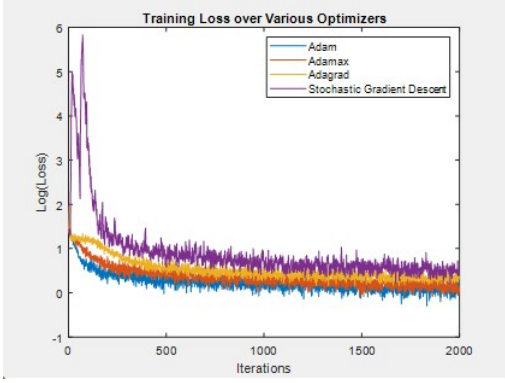

Figure 5. Training loss for 2000 iterations of various optimizers

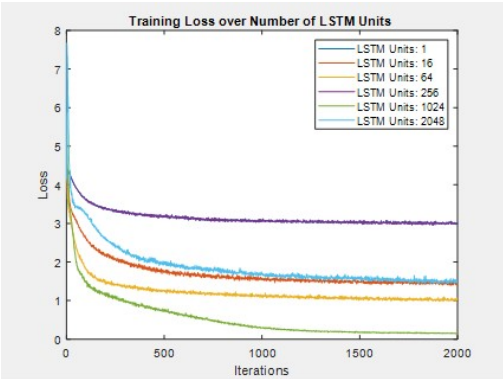

Figure 6. Training loss for 2000 iterations of LSTM units

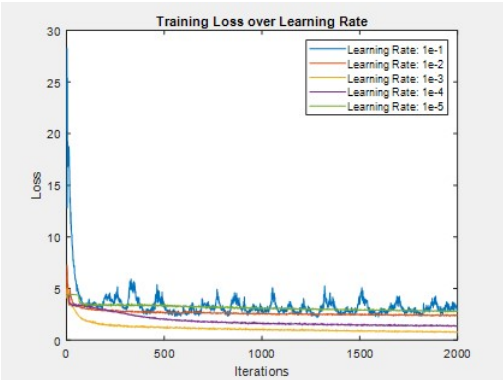

Figure 7. Training loss for 2000 iterations of the learning rate

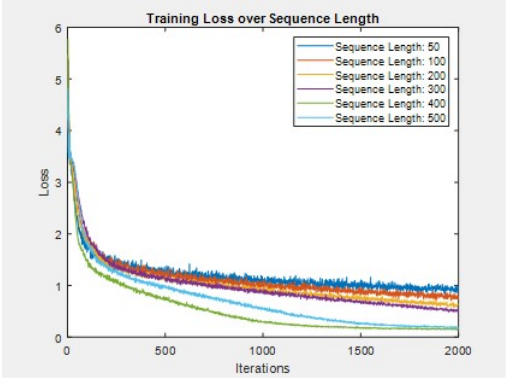

Figure 8. Training loss for 2000 iterations of the sequence length

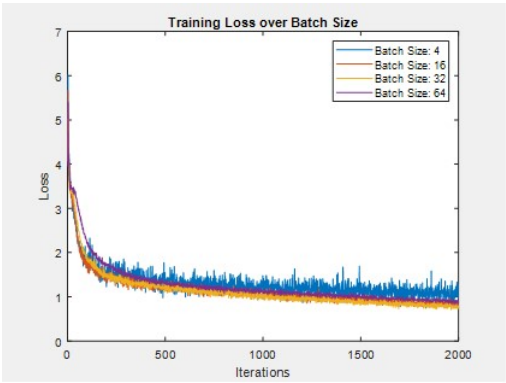

Figure 9. Training loss for 2000 iterations of the batchsize

Parameter	Optimal Value
Optimizer	Adam
LSTM Units	1024
Learning Rate	1e-3
Sequence Length	400
Batch Size	32

Table 1. Table of optimal parameters to be used in the final music generation

5. Conclusions

The paper here presents a very accessible solution to the problem of music generation. Using LSTM networks as a basis, the network architecture could be designed. However, there is much improvement that can be done in future work. For example, different loss functions and other measures of comparison should be evaluated. As mentioned in the Section 2, metrics like the Inception Score and the FID score are popular and successfully choices of the moment. Additionally, the prevent the over-fitting of a network trained to a specific and unvaried data set, data augmentation should be investigated. This could be achieved by adding various types of noise to the current data set, or by adding entirely new and diverse samples to it. Finally, a more structurally advanced network would ideally be able to produce harmonies and lyrics as well. All of these improvements can be preformed on the existing network structure presented in this paper.

References

[1] Amini, Alexandar. "MIT 6.S191: Introduction to Deep Learning." github.com. Jan 2020. Web. ¡https://github.com/aamini/introtodeeplearning¿. 2

[2] Banitalebi-Dehkordi, Mehdi, and Amn Banitalebi-Dehkordi. "Music Genre Classification using Spectral Analysis and Sparse Representation of the Signals." Journal of Signal Processing Systems (2018)Web. 2

[3] Briot, Jean-Pierre, Gaëtan Hadjeres, and François Pachet. "Deep Learning Techniques for Music Generation-A Survey." arXiv preprint arXiv:1709.01620 (2017). 1

[4] Chollet, Francois. "Keras API." keras.io. Mar 2019. Web. ¡https://keras.io/guides/functional-api/¿. 4

[5] Cohn, Eliza. "Deep Learning Final Repo." github.com. May 10, 2020. Web. ¡https://github.com/ecohn44/DeepLearningFinal¿. 4

[6] Dong, Hao-Wen, and et al. "MuseGAN: Multi-Track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment".2017. Print. 2

[7] Goodfellow, Ian, and et al. Deep Learning. MIT Press, 2016. Print. 3

[8] "Google Magenta." Nov, 2018. Web. ¡https://magenta.tensorflow.org/¿. 2

[9] Herremans, Dorien, Ching-Hua Chuan, and Elaine Chew. "A Functional Taxonomy of Music Generation Systems." ACM Computing Surveys 50.5 (2017): 1–30. 2

[10] Heusal, Martin, and et al. "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium." Neural Information Processing Systems (2017)Web. 2

[11] Kotecha, Nikhil, and Paul Young. Generating Music using an LSTM Network., 2018. Print. 1

[12] A. Lovelace. 1843. 'Notes on L. Menabrea's 'Sketch of the Analytical Engine Invented by Charles Babbage, Esq."' Taylor's Scientific Memoirs 3 (1843) 1

[13] Larochelle, Hugo, and et al. "Exploring Strategies for Training Deep Neural Networks." Journal of Machine Learning Research (2009)Web. 1

[14] Payne, Christine. "MuseNet." OpenAI, 25 Apr. 2019, openai.com/blog/musenet 2

[15] R.C. Pinkerton. 1956. Information theory and melody. Scientific American 194, 2 (1956), 77–86. 2

[16] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. In NIPS, 2016. 2

[17] Sherstinsky, Alex. "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network." Physica D: Nonlinear Phenomena 404 (2020): 132306. Crossref. Web. 1

[18] Walshaw, Chris. "ABC Notation." abcnotation.com. Dec 2011. Web. ¡http://abcnotation.com/software¿. 3

[19] Walshaw, Chris. "ABC Music Notation: Introduction". abcnotation.com. Retrieved March 1, 2008. 2