Automatic Music Generation using Long Short-Term Memory Neural Networks

Mannmay Vinze[1], Jeevan Danve[2], Srijan Shankar[3], Meraz Khan[4], Nilima Kulkarni[5]
[1,2,3,4,5]*Department of Computer Science and Engineering, MIT School of Engineering, MIT Arts Design and Technology University, Pune, 412201, India*

Abstract **- Recent developments in neural networks and sequential models has produced state of the art results in eld signal processing and sequential data generation. For us, music is a pleasing sound, and everyone listens to music very frequently, but computers represent it as an sequential data and all sequential data generation model can be used to generate music. Our work focuses on generating music using LSTMs. LSTMs have good capability to remember previous data, they have memory unlike general RNNs. A good music must not be abruptly changing tones and themes, it must be consistent and for this purpose our model must remember what was generated previously. So, LSTMs are the perfect choice for this context-based data generation. We used Keras[2], an open-source software library that provides a Python interface for artificial neural networks, to create and train model. Most impressive results were produced by Multi-layered Char-RNN with LSTM Cell.**

The data is represented with ABC le format for easier access and better understanding. We preprocess the data to make it more robust and understandable for neural network and decode it back for human interpretation, the preprocessing algorithms and data representation is thoroughly discussed. The model used in this paper learn the sequences of polyphonic musical notes over a Stacked-Multilayered Char-RNN with LSTM cell. The model required and do have have the potential to recall past details of a musical sequence and its structure for better learning because of memory cells present in LSTM cells. The whole architecture with data ow and training and testing scores are explained.

1.INTRODUCTION

This paper focuses on generating music automatically using Sequence to Sequence Recurrent Neural Network (RNN) Models [4].

Even someone who is not a musician but good at deep learning can try to generate quality music using RNN. Our task here is to train some neural network models to generate music. The model will lean to produce human pleasing sound from training data provided to it. We also expect our model add some innovation to the music, we do not want it to create some already created music because that will be just equivalent to remixing the music, which surely is not the task here. So, our aim is to produce original and consistent music.

1.1Recurrent Neural Networks

Recurrent Neural Networks are a category of Sequential Models, they work on sequential data/time series data. Since music can be represented as a sequential signal in time for computer, sequential models are perfect choice to work on music. RNNs use concept of weight sharing over time, they will use same weights/cell for each timestep. The general architecture of RNN and how it unfolds in time is represent in Figure 1.

RNNs have many variants and one of the most used variant of RNN is Long Short-Term Memory Network (LSTM)[1]. A general RNN does not have any memory cell and performs bad in long dependency tasks, LSTM resolve this problem by introducing memory cells. RNNs also su er with vanishing gradient problems, this problem is also resolved by LSTM.

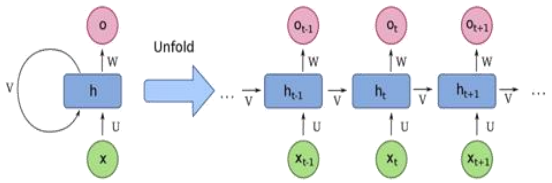


Figure 1: Unfolding of Recurrent Neural Network with each timestep. [from Reference 1: [1] - Sepp Hochreiter, Jurgen Schmidhuber. LONG SHORT-TERM MEMORY. Neural Computation 9(8):1735-1780, 1997]

1.2 Long Short-Term Memory Networks

LSTM [1] was originally proposed by Sepp Hochreiter, Jurgen Schmidhuber in 1997, a lot theoretical and experimental works have been published on the subject of this type of an RNN, many of them produced astounding results achieved across a wide variety of application domains which involved sequential data. The impact of the LSTM network has been quite signi cant in language modeling, speech-to-text transcription, machine translation, and other applications which involve time series or sequential data. The LSTM resolve problem of vanishing gradient and handles long term dependencies by introducing memory cells. The architecture of a general LSTM Cell is described in gure 2.

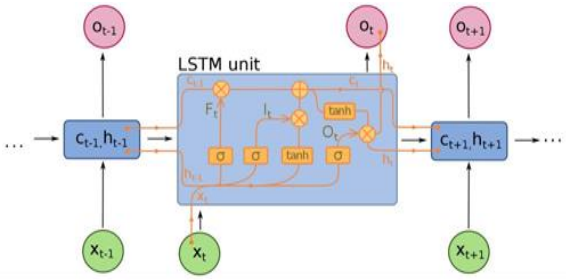


Figure 2: LSTM Cell

[from Reference 1: [1] - Sepp Hochreiter, Jurgen Schmidhuber. LONG SHORT-TERM MEMORY. Neural Computation 9(8):1735-1780, 1997]

LSTM is augmented by recurrent gates known as \forget gates". LSTM, as mentioned, prevents backpropagated errors from vanishing or exploding. Instead, errors can ow backwards through unlimited numbers of virtual layers unfolded in space. That is, LSTM can learn tasks that require memories of events that happened thousands or even millions of discrete time steps earlier. LSTM works very well with long delays between signi can't events and can handle signals that mix low and high frequency components.

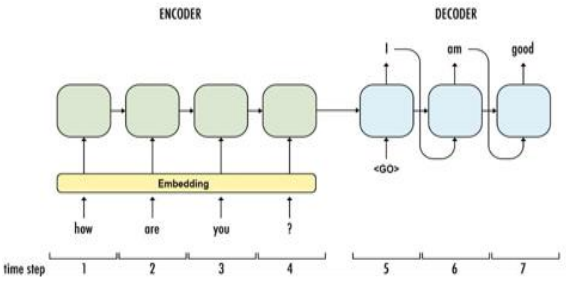


Figure 3: Sequence to Sequence Model [from Reference 5: [5] - Cedric De Boom, Thomas Demeester, Bart Dhoedt. Character-level Recurrent Neural Networks in Practice: Comparing Training and Sampling Schemes]

1.3 Sequence to Sequence Models

Generally, when dealing with sequential models, there are three possible input-output combinations. First, input is a sequence, and we need to predict some score or one scalar/categorical value. Second, input is a sequence and output are also a sequence, popularly known as Sequence to Sequence (Seq2Seq) models [4]. Third, input is a scalar and output are a scalar.

Our use case involves second model, Sequence to Sequence model. As the name suggests, we feed a sequence to the model and the model produces equivalents sequence in other domain (like language translation) or extrapolates/generate next part of the sequence. Figure 3 represents a normal sequence to sequence model for language translation.

1.4 Char-RNN Model

The char-RNN [5] produces the output character by character. It predicts the probability of next character out of all possible characters based on previously produced character. We can represent each tone of music as a character a feed to it. Figure 4 show architecture of char RNN.

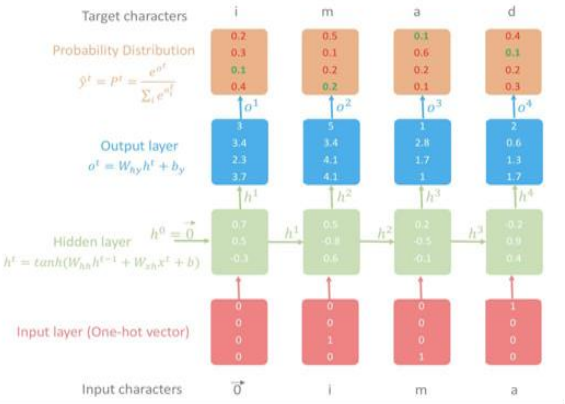


Figure 4: Architecture of Char RNN

[from Reference 6 –[6]- Zhiyong Cui, Ruimin Ke, Ziyuan Pu, Yinhai Wang. Stacked Bidirectional and Unidirectional LSTM Recurrent Neural Network for Network-wide Traffic Speed Prediction]

2.METHOD

Now, we discuss the preprocessing of data and model architecture. There are various ways to represent the music (ABC, MIDI). But here we are using char RNN

for music generation, so our input or representation should be compatible with our model architecture. So, we used ABC representation of music, it uses characters to represent tones of music, which the format of data accepted by Char-RNN Architecture. After completion of prediction, we convert the data to MIDI format and then to an mp3 le.

2.1Input Data

The data is taken from ABC Version of Nottingham Music Database 1. We trained our model on Jigs and Hornpipes datasets from all datasets available on the collection. The jigs dataset contains 340 tunes and Hornpipes data contains 60 tunes. So, there are 400 datapoints in our datasets.

2.2 Data Preprocessing

The data is fed into model in batches to prevent memory over ow. We will feed batch of sequences at once into our model.

Following are the details of one batch:

Batch Size = 16

Sequence Length = 64

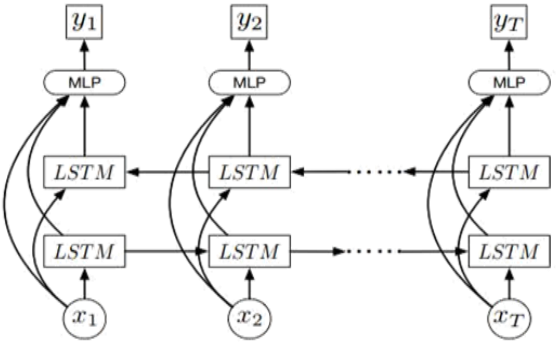


Figure 5: Stacked LSTM layer

There are total of 155222 characters in our data and total number of unique characters are 87. We have assigned a numerical index to each unique character. We have created a dictionary where key belongs to a character and its value is it's index.

2.3 Model Architecture and Training

As described in figure 4 we have stacked RNN and then using output of last RNN layer we produce probability distribution for next character. We use LSTM RNN so it can handle long term dependencies and produce consistent and pleasant music. Our model contains following layers.

2.3.1 Stacked LSTM Layer

Instead of using one LSTM cell, we use stacked LSTMs, which means we send output of a LSTM to another LSTM and do it few times and then send it generate probablities. Figure 5 describes architecture of stacked LSTM.

2.3.2 Dense Layer

Dense layer is a layer of fully connected neural networks. Dense layer works on output of stacked LSTM layer.

2.3.3 Softmax Layer

Softmax layer assigns decimal probabilities to each class in a multi-class problem. Those decimal probabilities add up to 1.0. This additional constraint helps training converge more quickly than it otherwise would. Figure 6 shows softmax layer and activation function used in it.

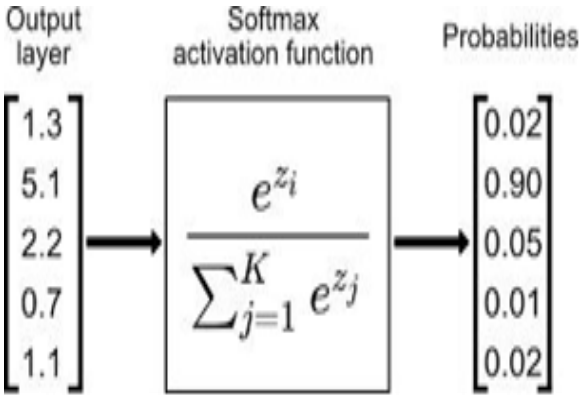


Figure 6: Softmax Layer

[Simpy a mathematical function representation]

2.3.4 Dropout Layer

The Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training time, which helps prevent overfitting. Inputs not set to 0 are scaled up by 1/ (1 - rate) such that the sum over all inputs is unchanged. It helps with regularization of model weights.

2.4 Music Generation

We trained the model with different combinations of hyperparameters and have weights of the one which produces best results. Now we are ready to generate music using that trained model. For music generation, we provide one of the 87 unique characters as input to the model and actually generate an 87 probability values using softmax layer. From these 87 probability values we achieved, we select the next attribute in a probabilistic and non-speci c way, and give the

selected attribute to the model for next prediction and repeat this process. This way we continue to combine output characters to produce music with different heights. And the end, after combining all outputs we get our music.

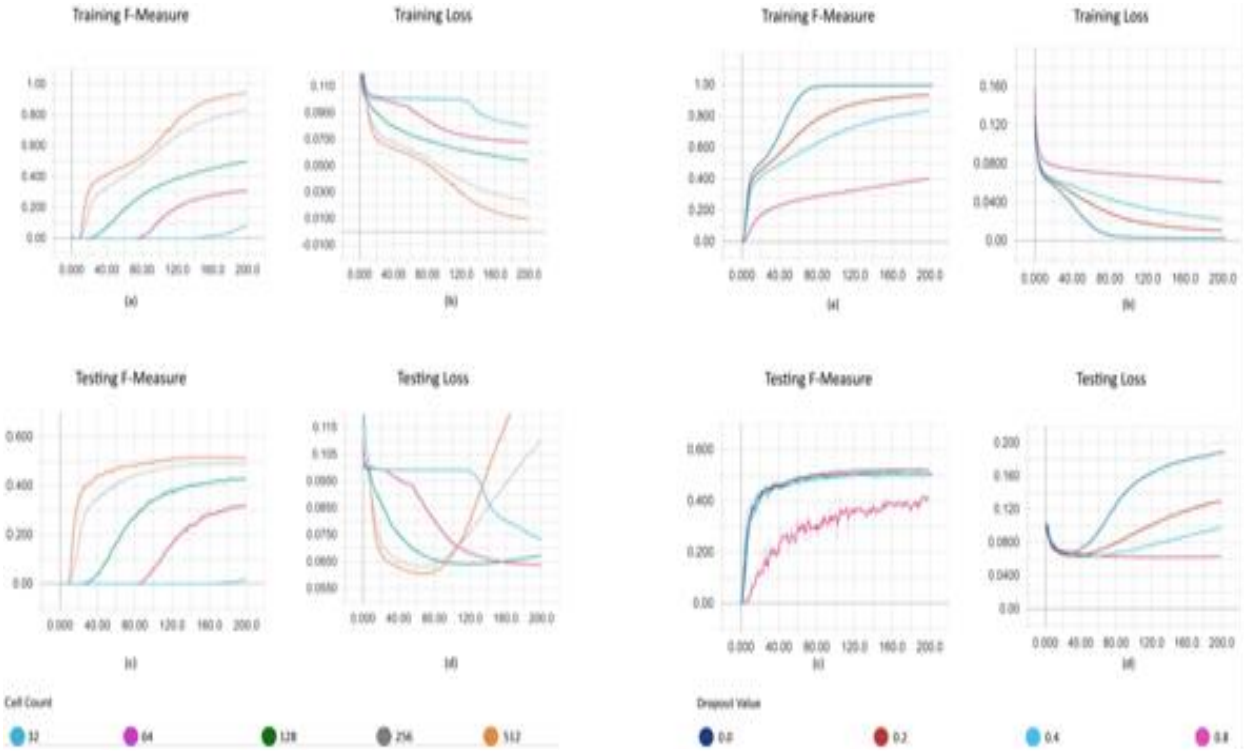
3 RESULTS

All the training and testing was done using Keras. We tried Adam, RMSProp, Stochastic Gradient, and Adagrad with learning rate 0.001. RMSProp worked best, the comparison was done on basis of F-Score. Number of training epochs were 200. The data was splitted with 70-30 split of train and validation sets.

It has been observed that having many LSTM layers made the learning progress slower and less accurate than having one or two layers with a sufficient number of neurons. This might be because of the 'Vanishing Gradient' problem where the depth of the network prevents the layers near the input to update their weights in an e effective manner. We might have been able to get a deeper network if we found a way to combat this problem. One solution we didn't get to try was using 'Leaky ReLu' activation functions. We ended up using LSTM layers in the range of 3{4 layers.

The size of the LSTM layers had a big impact on the result. The bigger the size, the faster it converged, however this came with overfitting. We expect this was because having more neurons permitted the network to save more of the data of the training set into the weights instead of optimizing a way to generalize the overall patterns of music.

Figure 7a shows training and testing graph with different LSTM layer sizes. Figure 7b shows training and testing graph with different dropout values.

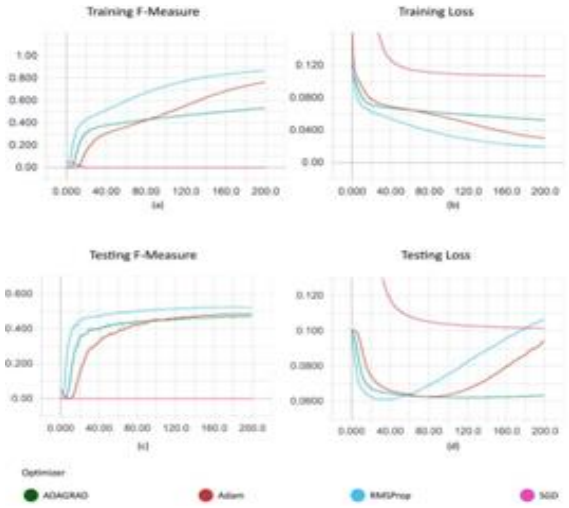


(a) F1 score and loss for different LSTM layer sizes. (b) F1 score and loss for different dropout values.

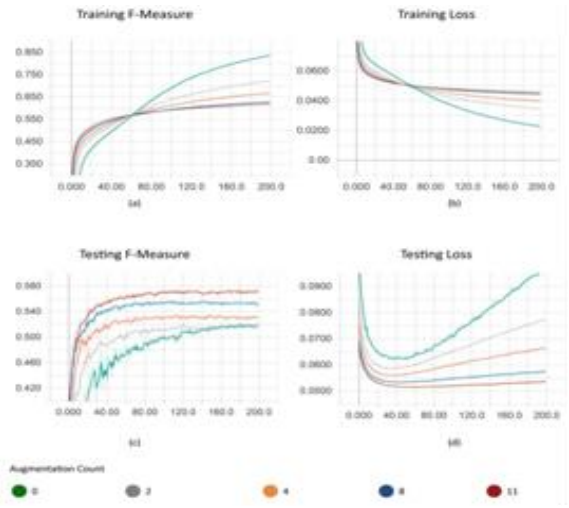
Figure 7: F1 score and loss for different LSTM layer sizes and dropout values.

Figure 8a shows training and testing graph with di erent optimizers. Figure 8b shows training and testing graph with di erent dropout values.

Figure 9a shows training and testing graph with di erent batch sizes. Figure 9b shows training and testing graph with di erent testing window sizes.

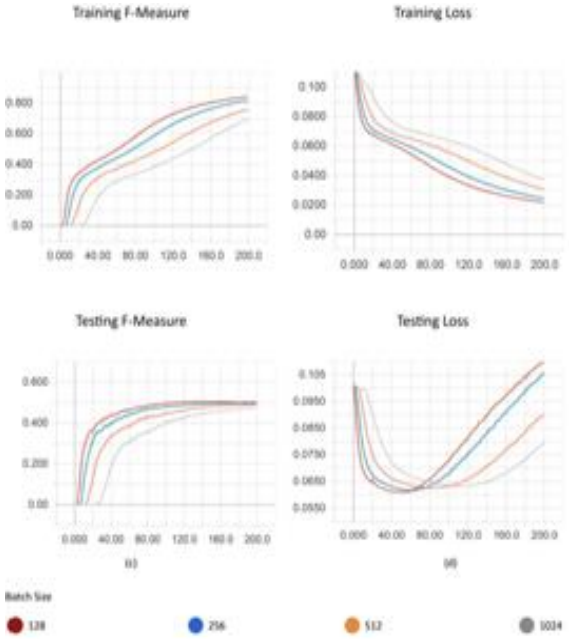


(a) F1 score and loss for different optimizers.

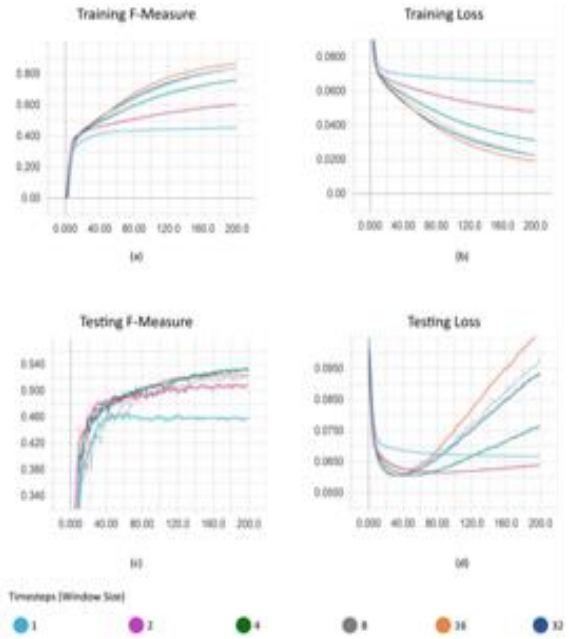


(b) F1 score and loss for different augmentation values.

Figure 8: F1 score and loss for different optimizers and augmentation values.



(a) F1 score and loss for different batch sizes.



(b) F1 score and loss for different testing window sizes.

Figure 9: F1 score and loss for different batch sizes and testing window sizes.

REFERENCE

[1] Sepp Hochreiter, Jurgen Schmidhuber. LONG SHORT-TERM MEMORY. Neural Computation 9(8):1735-1780, 1997

[2] Chollet, F. & others, 2015. Keras. Available at: https://github.com/fchollet/keras.

[3] Sanidhya Mangal, Rahul Modak, Poorva Joshi. LSTM Based Music Generation System.

[4] Ilya Sutskever, Oriol Vinyals, Quoc V. Le. Sequence to Sequence Learning with Neural Networks.

[5] Cedric De Boom, Thomas Demeester, Bart Dhoedt. Character-level Recurrent Neural Networks in Practice: Comparing Training and Sampling Schemes

[6] Zhiyong Cui, Ruimin Ke, Ziyuan Pu, Yinhai Wang. Stacked Bidirectional and Unidirectional LSTM Recurrent Neural Network for Network-wide Traffic Speed Prediction