Design and Development of Music Composition System

Tanu Khandate[1] Deepti Ullikashi[2] Deepa Jadhav[3] Prof. J V Vadavi[4]
[1,2,3,4]Department of Computer Science and Engineering
[1,2,3,4]SDM College of Engineering and Technology Dharwad, India

Abstract— In general music composed by recurrent neural networks (RNNs) suffers from a lack of global structure. Though networks can learn note-by-note transition probabilities and even reproduce phrases, attempts at learning an entire musical form and using that knowledge to guide composition have been unsuccessful. The reason for this failure seems to be that RNNs cannot keep track of temporally distant events that indicate global music structure. Long Short-Term Memory (LSTM) has succeeded in similar domains where other RNNs have failed, such as timing and counting and CSL learning. In the current study we show that LSTM is also a good mechanism for learning to compose music. We compare this approach to previous attempts, with particular focus on issues of data representation. We present experimental results showing that LSTM successfully learns a form of piano music and is able to compose novel (and we believe pleasing) melodies in that style. Remarkably, once the network has found the relevant structure it does not drift from it: LSTM is able to play the piano with good timing and proper structure as long as one is willing to listen.

Keywords: RNN (Recurrent Neural Network), LSTM (Long Short Term Memory), MIDI (Musical Instrument Digital Inter- face), Char RNN Model, Music21

I. INTRODUCTION

The most straight-forward way to compose1 music with an RNN is to use the network as single-step predictor. The network learns to predict notes at time t+1 using notes at time t as inputs. After learning has been stopped the network can be seeded with initial input values- perhaps from training data-and can then generate novel compositions by using its own outputs to generate subsequent inputs. A feed-forward network would have no chance of composing music in this fashion. Lacking the ability to store any information about the past, such a network would be unable to keep track of where it is in a song. In principle and RNN does not suffer from this limitation. With recurrent connections it can use hidden layer activations as memory and thus is capable of exhibiting (seemingly arbitrary) temporal dynamics. In practice, however, RNNs do not perform very well at this task. As Mozer (1994) aptly wrote about his attempts to compose music with RNNs, "While the local contours made sense, the pieces were not musically coherent, lacking thematic structure and having minimal phrase structure and rhythmic organization". The reason for this failure is likely linked to the problem of vanishing gradients (Hochreiter et al., 2001) in RNNs. In gradient methods such as Back-Propagation through Time (BPTT) (Williams and Zipser, 1995) and Real-Time Recurrent Learning (RTRL) (Robinson and Fallside, 1987) error flow either vanishes quickly or explodes exponentially, making it impossible for the networks to deal correctly with long-term dependencies.

II. SYSTEM ARCHITECTURE

Complete system architecture has been represented below.

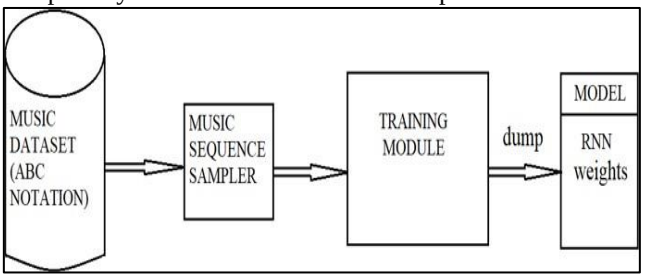


Fig. 1: TRAINING PHASE

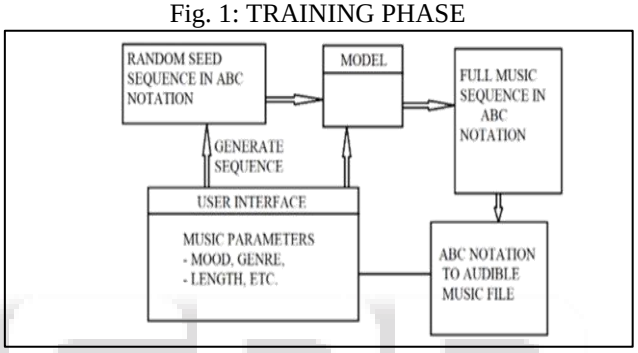


Fig. 2: APPLICATION PHASE

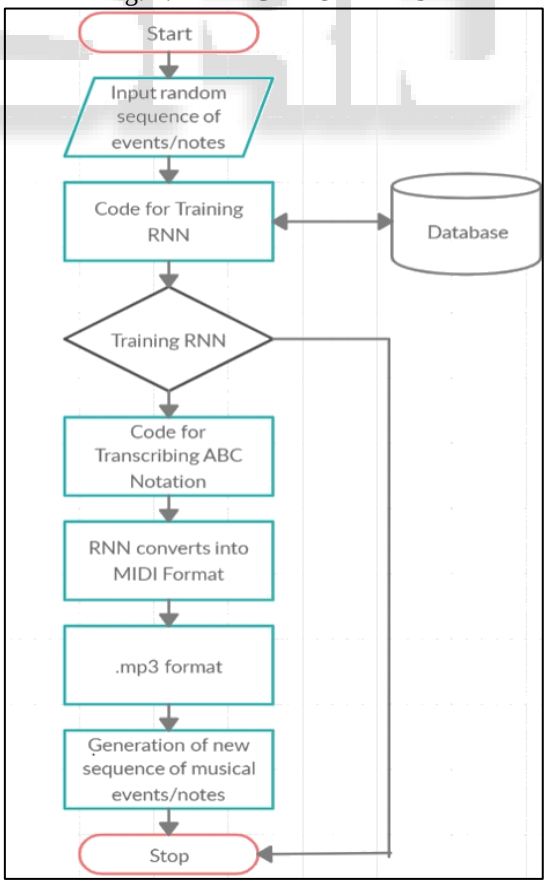


Fig. 3: FLOW CHART

All rights reserved by www.ijsrd.com

III. DATASET DESCRIPTION

A. Nottingham Dataset:

- This is a collection of 1200 British and American folk tunes, (hornpipe, jigs, and etc.) that was created by Er Foxley. The database was converted to abc music notati format.
- From data-source, we have downloaded two files:
  Jigs (340 tunes)
  Hornpipes (65 tunes)
- Dataset will be in this form:

```
X: 1
T:A and D
% Nottingham Music Database
S:EF
M:4/4
K:A
M:6/8
P:A
f|"A"ecc c2f|"A"ecc c2f|"A"ecc c2f|"Bm"BcB "E7"B2f|
"A"ecc c2f|"A"ecc c2c/2d/2|"D"efe "E7"dcB| [1"A"Ace a2:|
 [2"A"Ace ag=g||\
K:D
P:B
"D"f2f Fdd|"D"AFA f2e/2f/2|"G"g2g ecd|"Em"efd "A7"cBA|
"D"f^ef dcd|"D"AFA f=ef|"G"gfg "A7"ABc |1"D"d3 d2e:|2"D"d3 d2||
```

Fig. 4: DATASET

- We have found out that there are total of 155222 characters in our data. Total number of unique characters are 87.
- We have assigned a numerical index to each unique char- acter. We have created a dictionary where key belongs to a character and its value is its index. We have also created an opposite of it, where key belongs to index and its value is its character.

How batches are constructed:

	Batch-1	Batch-2	...	Batch-150	Batch-151
0	0...63	64...127	...	9536...9599	9600...9663
1	9701...9764	9765...9829	...	19237...19300	19301...19364
.	...	...	...	...	...
.	...	...	...	...	...
.	...	...	...	...	...
14	135814...135877	135878...135941	...	145350...145413	145414...145477
15	145515...145578	145579...145642	...	155051...155114	155115...155178

***There are total of 155222 characters in our dataset, out of which only 87 characters are unique. We have assigned index to each of the character. So, the above numbers in the batches are not the exact numbers. In reality the batches will contain index of the corresponding character.*

Fig. 5: CONSTRUCTION OF BATCHES

IV. PROPOSED ALGORITHMS

A. Algorithm1: Recurrent Neural Network (RNN)

- We will feed data into batches. We will feed batch of sequences at once into our RNN model. First we have to construct our batches.
- We have set following parameters: Batch Size = 16
  Sequence Length = 64

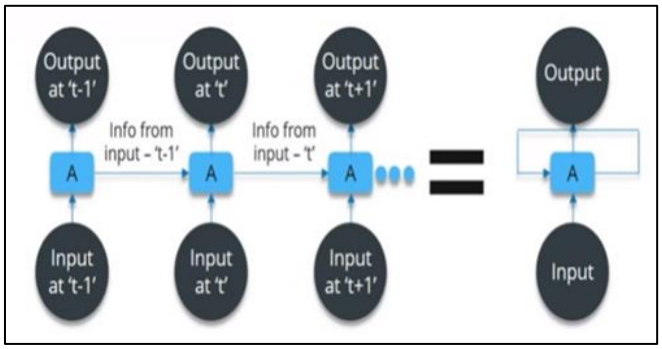


Fig. 6: RNN FLOW DIAGRAM

RNN as they are called in short, are a very important variant of neural networks heavily used in Natural Language Processing. In a general neural network, an input is processed through a number of layers and an output is produced, with an assumption that two successive inputs are independent of each other. This assumption is however not true in a number of real-life scenarios. For instance, if one wants to predict the price of a stock at a given time or wants to predict the next word in a sequence it is imperative that dependence on previous observations is considered.

RNNs are called recurrent because they perform the same task for every element of a sequence, with the output being depended on the previous computations. Another way to think about RNNs is that they have a "memory" which captures information about what has been calculated so far. RNN has disadvantages:

1) Gradient vanishing and exploding problems.
2) Training an RNN is a very difficult task.
3) It cannot process very long sequences if using tanh as an activation function. To overcome all these disadvantages, we use Long Short Term Networks (LSTM's).

CharRNN

This code implements Single-layer Recurrent Neural Network (RNN, LSTM, and GRU) for training/sampling from character-level language models. In other words the model takes one text file as input and trains a Recurrent Neural Network that learns to predict the next character in a sequence. The RNN can then be used to generate text character by character that will look like the original training data.

Now our music is a sequence of characters. We will feed one after the other character of the sequence to RNN and the output will be the next character in the sequence. So, therefore, the number of output will be equal to the number of inputs. Hence, we will be using Many-to-Many RNN, where number of output is equal to the number of input. A more detailed image of RNN is shown below.

All rights reserved by *www.ijsrd.com*

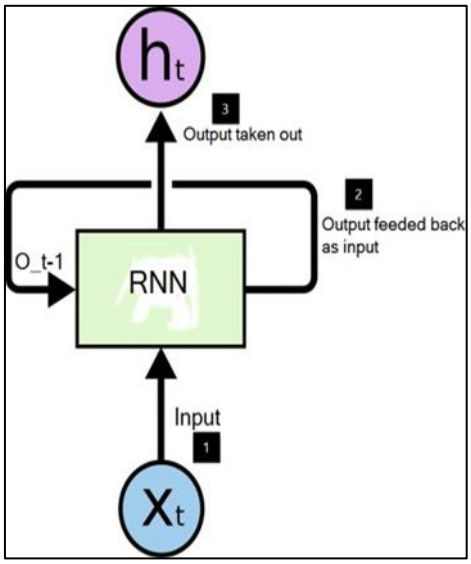


Fig. 7. RNN

In the above image, Xt is a single character at time‘t’ which is given as an input to RNN unit. Here, output of the previous time‘t-1’ character which is also given as an input to RNN at time‘t’. It then generates output ‘ht’. This output ‘ht’ will again be given as input to RNN in next time step.

This output ‘ht’ will be a next character in sequence. Let say our music is represented as [a, b, c, a, d, f, e,. . .]. Now,

We will give first character ‘a’ as an input and expects RNN to generate ‘b’ as an output. Then in next time-step we will give ‘b’ as an input and expects ‘c’ as and output. Then in next time-step we will give ‘c’ as an input and expects ‘a’ as an output and so on. We will train our RNN such that it should output next character in the sequence. This is how it will learn whole sequence and generate new sequence on its own.

This will keep on going till we feed all of our inputs. Since, our music is a combination of many characters and our output is one of those characters so it can be thought of as multi-class classification problem. Here, we will use “Categorical Cross- Entropy” as a loss function which is also known as “Multi- Class log-loss”. In the last layer we will keep “Softmax” activations. The number of “Softmax” activation units in last layer will be equal to the number of all unique characters in all of the music in train data. Each RNN can be a LSTM which contains ‘tanh’ activation unit at – input-gate – which is a differentiable function. Therefore, this RNN structure can be trained using back-propagation and we keep on iterating it using “Adam” optimizer till we converge. At the end, our RNN will be able to learn sequence and patterns of all the musical notes that are given to it as input during training.

Once our char RNN model is trained, we will then give any one random character — from the set of unique characters that we feed to our char RNN during training time — to our trained char RNN, it will then generate characters automatically which will be based on the sequences and patterns that it has learnt during training phase.

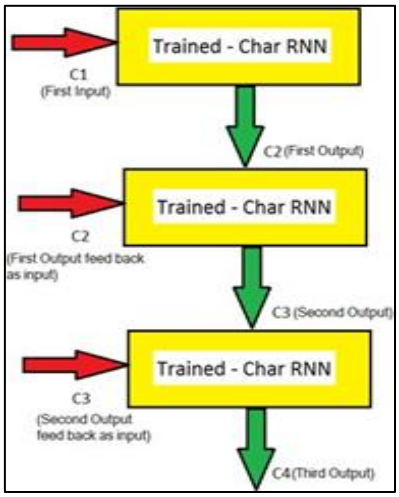


Fig. 8. GENERATE OUTPUT SEQUENCES

In the above, image we give “C1” as the input to our trained RNN. Note, that “C1” is a character which should be present in the set of the characters that we feed to our char RNN during training time. Now our trained char RNN will generate output “C2”. This “C2” output is feedback as input again to the trained char RNN. This will generate “C3” as an output. This “C3” output is feedback as input again to the trained char RNN and so on. Now we got new sequence of music [C1, C2, C3. . .]. This new characters of sequence is a new music generated by our trained char RNN which is based on the sequences and patterns that it has learnt during training phase.

B. Algorithm2: Long Short Term Memory (LSTM’s)

They are special kind of RNN’s.

They are capable of learning long term dependencies. LSTM’s have a Nature of Remembering information for a long periods of time is their Default behavior.

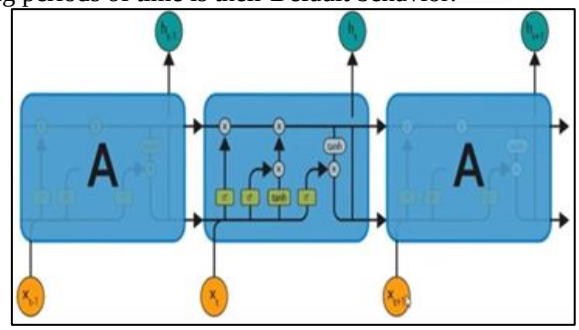


Fig. 9. LSTM FLOW DIAGRAM

LSTM has four step process:

1) Step-1 The first step in the LSTM is to identify that information that are not required and will be thrown away from the cell state. This decision is made by a sigmoid layer called as forget gate layer.
   ft=(wf[ht-1,xt]+bf) wf = Weight
   ht-1= Output from the previous time stamp xt = New input
   bf = Bias
2) Step-2 The next step is to decide, what new information we’re going to store in the cell state. This whole process comprises of following steps. A sigmoid layer called the “input gate layer” decides which values will be

All rights reserved by *www.ijsrd.com*

updated. Next, a tanh layer creates a vector of new candidate values that could be added to the state.
it=(wi[ht-1,xt]+bi) ct=tanh(wc[ht-1,xt]+bc)
In the next step, we'll combine these two to update the state.

3) Step-3 Now, we will update the old cell state, Ct-1, into the new cell state Ct. first, we multiply the old state (Ct-1) by ft, forgetting the things we decided to forget earlier. Then, we add it*ct. This is the new candidate values, scaled by how much we decided to update each state value. ct=ft*ct-1+it*ct
4) Step-4 We will run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through tanh (push the values to be between -1 and and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to. ot=(wo[ht-1,xt]+bo) ht=ot*tanh(ct)

1) Phase1: The current goal is to learn to predict music sequences on the Nottingham Music Dataset which contains music scores in ABC Notation. To parse the ABC Notation music files into notes we use music21 which allows converting ABC Notation into individual notes and also in rendering the music via MIDI file.
2) Phase2: First we collect use the parsers in music21 to collect all the unique notes in the dataset and use this as the set of tokens and create a sequence of one-hot encoding of the music sequences from which we sample input and target sequences. Note that the target sequence is ahead of the input by just one step (note).
3) Phase3: The CharRNN model consists of either Gated Recurrent Units (GRUs) or Long Short Term Memory (LSTM) Units (with possibly multiple layers) and with Dropout Normalization for each layer. The CharRNN model predicts probabilities for each class i.e each note in the unique set. In Keras library, in LSTM there is a parameter called 'return sequences'. It is False by default. But if it is True, then each RNN unit will generate output for each character means at each time step. This is what we want here.
The loss optimized is the cross entropy loss between the predicted class probabilities and the one-hot target output sequence.
4) Phase4: We have applied 'TimeDistributed' dense layers with "Softmax" activations in it. This wrapper applies a layer to every temporal slice of an input.
We have 87 unique characters in our dataset and we want that the output at each time-stamp will be a next character in the sequence which is one of the 87 characters. So, time- distributed dense layer contains 87 "Softmax" activations and it creates a dense connection at each time-stamp. Finally, it will generate 87 dimensional output at each time-stamp which will be equivalent to 87 probability values. It helps us to maintain Many-to-Many relationship. The output sequence of notes so obtained is converted to MIDI format using music21.

1) Explanation of Many-to-Many RNN

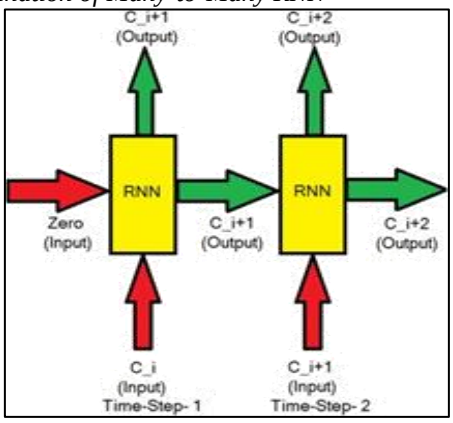


Fig. 10: MANY-TO-MANY RNN

Above image shows Many-to-Many RNN. The yellow box is a single RNN unit where 'Ci' is a first character input given to RNN unit. For first RNN we also have to provide zero input which is nothing but a dummy input because RNN always takes current input and previous output as input. Since, we don't have previous output for the first iteration so we input zero, only for first iteration. Now, we want our RNN to produce next character as output. So, the output of 'Ci' will be 'Ci+1' which is nothing but a next character in sequence. Now, our next input itself is a next character, and at the same time the output of previous input is also next character so we feed both of them to next RNN unit again and produces 'Ci+2' as output which is next character in the sequence. This is how Many-to-Many RNN works. Note, the above image shows time-unwrapping of RNN. It is only one RNN unit which repeats itself at every time-step.

Time-distributed dense layer contains 87 "Softmax" activations and it creates a dense connection at each time-stamp. Finally, it will generate 87 dimensional output at each time-stamp which will be equivalent to 87 probability values. It helps us to maintain Many-to-Many relationship.

There is one more parameter in LSTM which is known as "stateful". Here, we have given "stateful = True". If True, then the last state for each sample at index 'i' in a batch will be used as initial state for the sample of index 'i' in the following batch. This is used because all of the batches contains rows in continuation. So, if we feed the last state of a batch as initial state in the next batch then our model will learn longer sequences.

2) Model Architecture

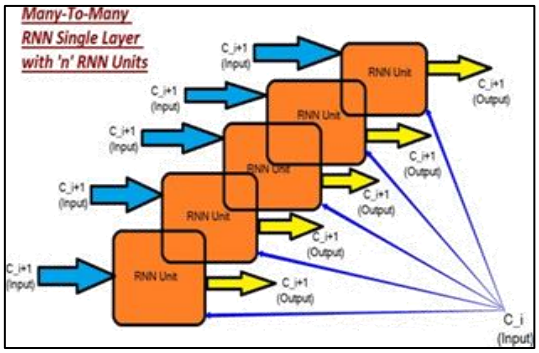


Fig. 11: MANY-TO-MANY RNN SINGLE LAYER WITH 'n' RNN UNITS

All rights reserved by *www.ijsrd.com*

The above image shows 'n' RNN units in single RNN layer. We have constructed our single RNN layer exactly like this where we have 256 LSTM-RNN Units in one layer of RNN. At each time step all of the RNN units generate output which will be an input to the next layer and also the same output will again be any input to the same RNN unit. Here, in our project each RNN unit is an LSTM unit. In Keras library, in LSTM there is a parameter called 'returnsequences'. It is False by default. But if it is True, then each RNN unit will generate output for each character means at each time step. This is what we want here. We want our RNN unit to generate output as the next character when given input as previous character in the sequence. We have stacked up many LSTM units here so that each unit will learn different aspect of the sequence and create a more robust model overall.

In above image we have just shown one RNN layer with 'n' units. We have three such RNN layers each having 256 LSTM units. The output of each LSTM unit will be an input to all of the LSTM units in next layer and so on.

After three such layers of RNN, we have applied 'TimeDis- tributed' dense layers with "Softmax" activations in it. This wrapper applies a layer to every temporal slice of an input. Since the shape of each output after third LSTM layer is (16*64*256). We have 87 unique characters in our dataset and we want that the output at each time-stamp will be a next character in the sequence which is one of the 87 characters. So,

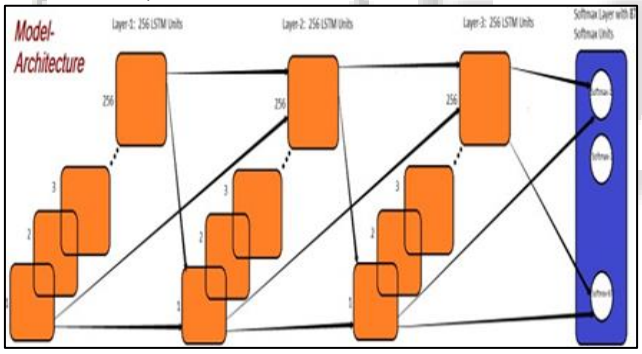


Fig. 12: MODEL ARCHITECTURE

Now we want to predict the next character which should be one of the 87 unique characters. So, it's a Multi-Class Classification problem. Therefore, our last layer is Softmax layer with 87 activations.

V. RESULTS

This is how music will be generated.

```
X: 1
T: Cooley's
M: 4/4
L: 1/8
R: reel
K: Emin
|:D2|EB{c}BA B2 EB|~B2 AB dBAG|FDAD BDAD|FDAD dAFD|
EBBA B2 EB|B2 AB defg|afe^c dBAF|DEFD E2:|
|:gf|eB B2 efge|eB B2 gedB|A2 FA DAFA|A2 FA defg|
eB B2 eBgB|eB B2 defg|afe^c dBAF|DEFD E2:|
```

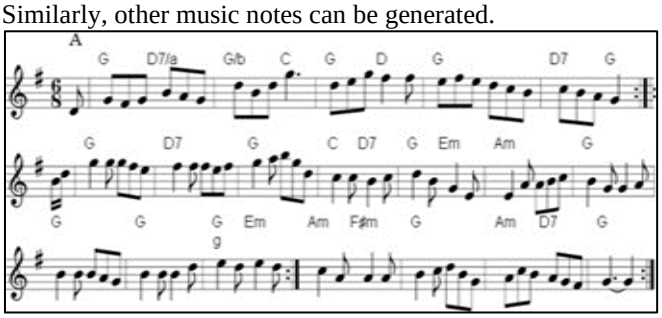
There are two parts in ABC-notation.

1) Part-1 represents Meta data. Lines in the Part-1 of the tune notation, beginning with a letter followed by a colon, indicate various aspects of the tune such as the index, when there are more than one tune in a file (X:), the title (T:), the time signature (M:), the default note length (L:), the type of tune (R:) and the key (K:).
2) Part-2 represents the tune, which is a sequence of characters where each character represents some musical note.

In our last model, we got only 82 percent accuracy. How- ever, in order to generate melodious music, we need at least 90 percent accuracy.

So, we have loaded the weights of last epoch from our previous model into this model and also we have added 2 extra layers of LSTM here with more LSTM units.

Here, we are fine-tuning our old layers and we have added more layers. In short, here we are doing"Transfer Learning" from old to new model.
Similarly, other music notes can be generated.



We have almost generated about 30 samples. Similarly, many such samples can be generated within the range of epoch weight, length of music sequence and characters, such that many combinations will be formed.

VI. FURTHER SCOPE

We have generated a good quality music, but there is a huge scope of improvement in it.

First, starting and ending music can be added in every new generated tune to give a tune a better start and better ending. By doing this, our generated music will become melodious.

Second, the model can be trained with more tunes. Here, we have trained our model with only 405 musical tunes. By training the model with more musical tunes, our model will not only expose to more variety of music but the number of classes will also increase. By this more melodious and at the same time more variety of music can be generated through the model.

Third, model can also be trained with multi-instrument tunes. As of now, the music generated is of only one piece of instrument. It would be interesting to listen what music the model will produce if it is trained on multi-instrument music.

Finally, a method can be added into the model which can handle unknown notes in the music. By filtering unknown notes and replacing them with known notes, model can generate more robust quality music.

VII. CONCLUSION

A music composition model based on LSTM successfully learned the global structure of a musical form, and used that information to compose new pieces in the form. Two

All rights reserved by *www.ijsrd.com*

experiments were performed. The first verified that LSTM was not relying on regularities in the melody to learn the chord structure. The second experiment explored the ability for LSTM to generate new instances of a musical form. These experiments are preliminary and much more work is warranted. However by demonstrating that an RNN can capture both the local structure of melody and the long-term structure of a musical style, these experiments represent an advance in neural network music composition.

REFERENCES

[1] Gers, F. A. and Schmidhuber, J. (2001). LSTM recurrent networks learn simple context free and context sensitive languages. IEEE Transactions on Neural Networks, 12(6):1333-1340.
[2] Hochreiter, S., Bengio, Y., Frasconi, P., and Schmidhuber, J. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In Kremer, S. C. and Kolen, J. F., editors, A Field Guide to Dynamical Recurrent Neural Networks. IEEE Press.
[3] Ian J. Goodfellow, Yoshua Bengio and Aaron Courville, "Deep Learn- ing", MIT Press, 2016.
[4] N Boulanger-Lewandowski, Y. Bengio, P. Vincent, Modeling Temporal Dependencies in High-Dimensional Sequences: Applications to Poly- phonic Music Generation and Transcription, in Proceedings of the 29th International Conference on Machine Learning (ICML), 2012.
[5] S. Sigtia, E. Benetos, S. Dixon, "An end-to-end neural network for polyphonic piano music transcription", IEEE/ACM Transactions on Audio Speech and Language Processing (TASLP), vol. 24, no. 5, pp. 927-939, 2016.
[6] I-Ting Liu and Bhiksha Ramakrishnan. Bach in 2014: Music com- position with recurrent neural network. Under review as a workshop contribution at ICLR 2015, 2015.
[7] M. Schedl, E. Go´mez, J. Urbano et al., "Music information retrieval: Recent developments and applications", Foundations and Trends@ in Information Retrieval, vol. 8, no. 2–3, pp. 127261, 2014.
[8] G. Peeters, "Chroma-based estimation of musical key from audio-signal analysis", ISMIR, pp. 115-120, 2006.
[9] Douglas Eck, Jurgen Schmidhuber, "A First Look at Music Composi- tion using LSTM Recurrent Neural Networks", Technical Report No. IDSIA-07-02, IDSIA / USI-SUPSI, Instituto Dalle Molle di studi sull' intelligenza artificiale, Galleria 2, CH-6900 Manno, Switzerland.
[10] Gaurav Sharma in Data Driven Investor, "Music Generation Using Deep Learning (A deep learning Case Study)", Oct17, 2018.
[11] Anirudh Rao a Research Analyst at Edureka, "Recurrent Neural Net- works (RNN) Tutorial, Analyzing Sequential Data Using TensorFlow in Python", May22, 2019.

All rights reserved by *www.ijsrd.com*