Music Generation Using LSTM Neural Network

Soumith Reddy Chinthalapally
University at Buffalo
Buffalo, New York, United States
soumithr@buffalo.edu

Abstract

There are many fields where Neural Networks are showing commendable grown with impeccable results matching humans in performance. Some of them are Image Recognition, Speech recognition, speech to text conversion. In this work I want to explore Music Generation using Character based RNN. There are already many kinds of implementations about music generation, I want to explore one kind of music generation which is least explored by others and demonstrate my findings.

1. Introduction

In this project I will use a Long Short-Term Memory (LSTM) network. Unlike standard feedforward neural networks, LSTM has feedback connections. It can not only process single data points but also entire sequences of data[2] which is used in this project.The task here is to take some existing music data then train the model using this existing data. The model has to learn the patterns in music that we humans enjoy. Once it learns these patterns, the model should be able to generate new music. It should not simply copy-paste from the training data instead it has to understand the patterns of music, to generate new music. We are not expecting the model to generate new music which is of professional quality, but we want it to generate a decent quality music which should be melodious and good to hear.

1.1. Motivation

After reading the article[9], it caught my attention and the implementation details looked interesting. I thought this will help me get a deep understand and hands on experience on LSTMs and RNN. After looking about more such articles I found that music can be represented in two forms, they are MIDI and abc format. Most of the experiments and research is done by generating using MIDI format, hence I thought to explore on abc format. The goal of the project is to generate unique melodious music similar to tunes from the "O'Neill's Music of Ireland", album of 1850 tunes on which the model is trained.

1.2. Input and Output

Input to the model is a sequence of musical events/notes. Output will be new sequence of musical events/notes. In this project I am taking single instrument music as input and generating a music of similar kind as output.

1.3. Data

The data is taken from opensource MIT website[1]. There are 1850 tunes which are in abc format individually. Each of the 1850 files are downloaded from the website using a script and put it in a single text file. We use this text file as input to the model.

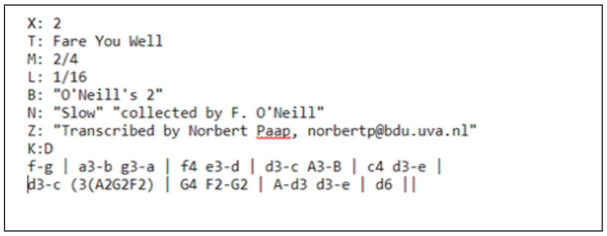
1.4. ABC Format

ABC notation is a shorthand form of musical notation. In basic form it uses the letters A through G, letter notation, to represent the given notes, with other elements used to place added value on these – sharp, flat, the length of the note, key, ornamentation[10]. There are two parts in abc format. 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:). Part-2 represents the tune, which is a sequence of characters where each character represents some musical note. Example of abc format is shown in the below Fig.1.

How the abc format text is represented in musical notes format for Fig.1 is shown in the Fig.2

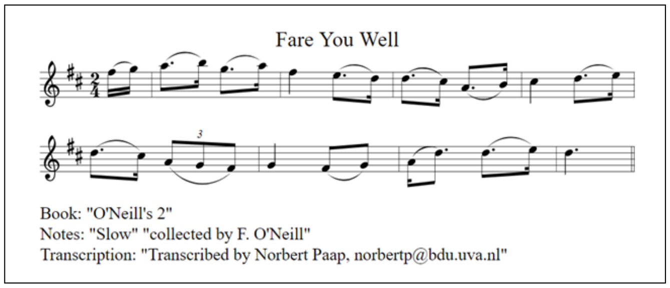
2. Background

Since the input music is a sequence of characters I will be using RNN or a variation of RNN which is good for understanding patterns in a sequential data input. I will be using CharRNN model which is a special type of RNN for the



```
X: 2
T: Fare You Well
M: 2/4
L: 1/16
B: "O'Neill's 2"
N: "Slow" "collected by F. O'Neill"
Z: "Transcribed by Norbert Paap, norbertp@bdu.uva.nl"
K:D
f-g | a3-b g3-a | f4 e3-d | d3-c A3-B | c4 d3-e |
d3-c (3(A2G2F2) | G4 F2-G2 | A-d3 d3-e | d6 ||
```

Figure 1. Sample abc format text



Fare You Well
3
Book: "O'Neill's 2"
Notes: "Slow" "collected by F. O'Neill"
Transcription: "Transcribed by Norbert Paap, norbertp@bdu.uva.nl"

Figure 2. Musical Notes representation of abc format text

model. We will feed a sequence of characters to the model and the model should try to predict the next character in the sequence. There are different variations of Char RNN like one-one ,one-many and many-many which are shown in Fig.3. We will be using many-many model where the model will try to predict output for each input value. The green blocks in the Fig.3 are RNN units which is a repeating structure which is shown in Fig.4.

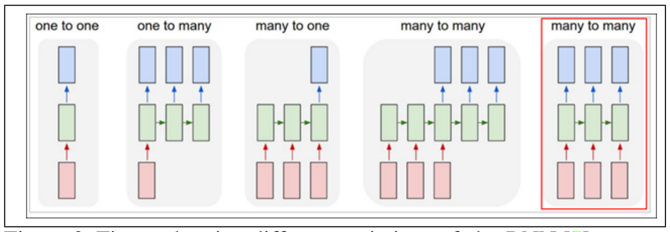


Figure 3. Figure showing different variations of charRNN [7]

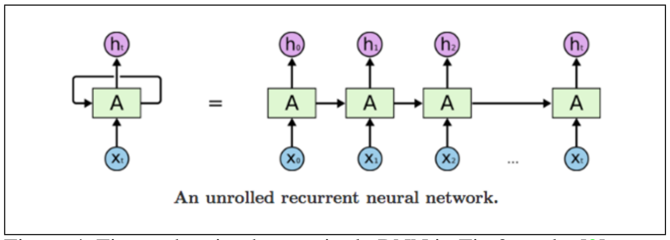


Figure 4. Figure showing how a single RNN in Fig.3 works [3]

Here the input we give is a single character Xt at time t, the feedback of the previous is added to the current layer. The output expected will be the next character in the input. 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 an 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 process will repeat until we feed all of our inputs. Since, our music is a combination of many characters and output is one of those characters so it can be thought of as multi-class classification problem.To calculate the loss for each iteration of the training we will be using categorical cross entropy since each of our outputs only belongs to a single class and we have a multi-class setting to work with. 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 the music in our training 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 until 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 model is trained, we will generate music. To generate music, we should give the initial random character from the set of random characters in our input to the model. The model will then generate characters automatically from the patterns which it learnt during training.The process of generating next character is shown in Fig.5.

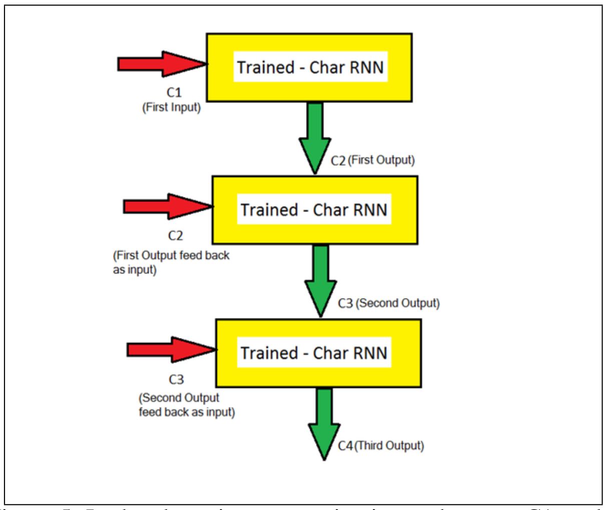


Figure 5. In the above image we give input character C1 to the model. The model will then predict the next character after C1 which is C2. We then send C2 as input to the next RNN which then predicts C3 and then C4 and the process goes on. Our final music sequence generated by our trained model is [C1,C2,C3. . . . .][8]

3. Model

3.1. Data Preparation

We have a text file as input to the model. The text files contains tunes of 1850 tunes.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
We have found out that there are total of 781989 characters in our initial data with total number of unique characters equal to 95.

While creating batches we generate X and Y from the input data where X is a 2D vector with index of characters stored the same order as they are in the file. Y is a 3D vector where it stores the corresponding next character of each X in a one-hot encoded manner. So for each value C(i) in X, Y store the one-hot encoding of C(i+1). Since we are using softmax classifier at the end, We want to make the problem a classification problem where the output can be any of the 87 unique characters.

3.2. Char RNN

We can have multiple LSTM cells between the input and output. The LSTMs together generate the output which is the next character. We have used 256 layers of LSTM in out project. For each layer we give C(i) as input, we get C(i+1) as output and we send C(i+1) as input to the next LSTM layer, which is clearly shown in Fig.6 below.

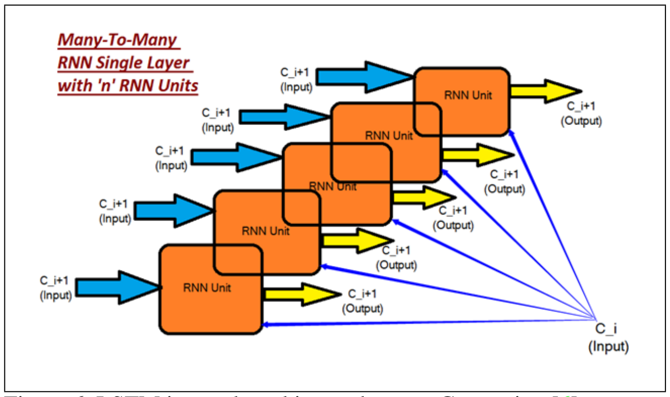


Figure 6. LSTM internal working and output Generation [6]

To generate one output for each input provided we use Return Sequences as ‘TRUE’ in the model definition. Since we are having return sequences as true and 256 LSTM units we will get a vector of size 256 as output for each input. For each timestep we take the 256 size vector and create a dense layer from it with the help of Time-Distributed Dense Layer of size 95(unique characters size). We give stateful as True for LSTM model because we are using batches and we want output the one batch as input to the next batch. For batch 1 the initial input is Zero but for next batch the output of current batch is given as input , because we are creating our batches continuously we want to give output of current batch as input to the next batch. This helps out RNNs to learn much longer sequences not just the sequence of a single batch. This helps our model to learn long term dependencies and generate much better output sequence.

3.3. Model Architecture

We are using a sequential model, where we have a embedding layer first to embed the characters to vectors of size 512. Since we know LSTMs take input only vectors and not characters, we are converting each unique character to a vector of size 512. We have three LSTM layers of size 256 as shown in the Fig.7 and for each LSTM layer we are adding a dropout with value of 0.2 to reduce overfitting. After the LSTM layers we are adding a Time Distributed dense layer and then Softmax activation layer. We are using Softmax because it is like a multi class classification task where our output can be any of the 95 classes(i. e the next character can be any of the 95 unique characters present in the data). The accuracy in the model context is if the model predicted the next character correctly then it can be said as accurate else it is called inaccurate. We haved used train_on_batch from keras to train the model on the batches we generated earlier.

3.4. Music Generation

Now we have a trained model we can start generating music. We will give an input to the trained model it should generate the probabilities of each of the 87 characters and give the character with highest probability as the next character. We will give C(i) as input we get C(i+1) as output and this is taken as input to generate C(i+2) and this process goes on continuously generating characters. In the last LSTM of our model while generating music we don’t give ”return_sequences = True” because here we will give only one character to generate the sequence. In the end, we just have to get one output which is equivalent to getting output at the last time-stamp. It is similar to one-to-one RNN, hence in last layer there is no need of giving return_sequences = True. To generate music we ask the user to provide the first character as input to the model and the number of character length music he wants typically from 300-600. The model will then generate music with the input character as first character and continuously generates characters till the length of the input given. If nothing is given as input it can take a random value as input and generate music.

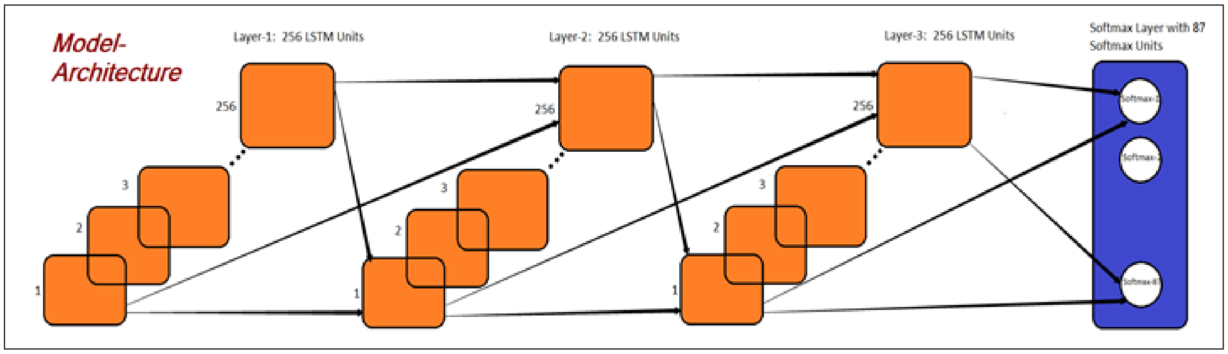


Figure 7. Model Architecture with 3 layers of 256 LSTMs units each [5]

4. Model Training

4.1. Training Process

Initially we take the input text file and get to know the number of unique characters in the file and then store it in a dictionary with each character as index and the value for each character is given from 0 to final character in a increasing order. We then dump this dictionary to a json file. We then divide the whole input text into Batches such that each batch is divided into 16 rows with 64 columns. Depending on the number of total characters in the input file we get different number of batches. For each batch X, which is a 2D array we store a respective Y, a 3D array which is one hot encoding of next character of each of X. Suppose C(i) is the index of character present in X[1,1] then one-hot encoding of C(i+1) next character of X is present at Y[1,1] which will be a vector of size equal to number of unique characters in the input text. We then send these X and Y values as parameters to model.train_on_batch function which returns loss and accuracy of the model. The model tries to predict Y for a given X and gets trained on it such that given a character it learns to predict the next character. I used early stopping to stop training if the loss does not decrease after 5 continuous epochs, this saves time and computation power. We store the weights of the model for each 10 epochs and store the best weights of the model in a .h5 file. We have used transfer learning approach where we used the previously stored best wights and send it to a model and train it again to increase the accuracy of the model. In the model we have an embedding layer which converts the input data to 512 length vector. Then we have 3 LSTM layers with 256 LSTM units in each layer. We then have a Time Distributed Dense layer and the output is sent to a softmax activation layer.

4.2. Music Generation

Once the model is trained, we can start generating music using the model. We will ask user to input the initial character from which user want to start generating your music and the length of the music the user want to generate. The model takes these as inputs and loads the best weights generated in the previous steps. We will take the initial character and send it to model.predict function which generates the probabilities of next character and then we generate the next character using np.random.choice() function giving the previously generated probabilities as input to the function. This generates the next character from the set of unique characters based on the generated probabilities associated with each entry in total unique characters.

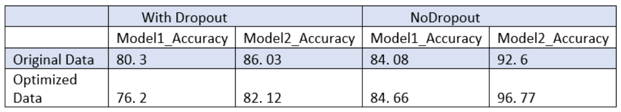
4.3. Accuracy

We are using accuracy as a metric to evaluate our model's performance. Accuracy in the model indicates how often the model is able to predict the correct next character given the current character. We will be passing the entire input text file to the model one character at a time, the model should predict next character for every character we send as input. If the character model predicted is the next character sent as input, then the prediction is accurate else it is said to be inaccurate. Accuracy is calculated as the ratio of total accurately predicted to the total predictions. A highly accurate model is one which can predict the next character given the current character which is helpful in generating a smooth music rather than a broken, discontinuous music.

5. Results

5.1. Initial Results

We have implemented the above architecture and got an accuracy around 80. To improve the accuracy we have used the transfer learning approach where the best weights obtained in the training is used to train the model again. We take the weights from initial model and load to a new model with similar architecture and train it again[4]. By doing this way the accuracy of the model increased to 86 which is better than previous model .



	With Dropout		NoDropout	
	Model1_Accuracy	Model2_Accuracy	Model1_Accuracy	Model2_Accuracy
Original Data	80. 3	86. 03	84. 08	92. 6
Optimized Data	76. 2	82. 12	84. 66	96. 77

Figure 8. In the above figure Model1 is the basic model and Model2 is the model that takes weights from Model1.

5.2. Data Processing

In our quest to increase the accuracy we observed that the data present is not completely required for music generation , there is some meta data in the data and we can remove the meta data from the text file which reduces the input data considerably. We observe that Title(T), Book(B), Notes(N) and Transciption(Z) is not required to generate music and removing them from our tunes does not change our original tune. Hence we are processing our data to remove those lines from all the 1850 tunes. This redundant data is anyways not useful for the model since we are focusing on generating music rather than the Title, Book or other info about the tune.

5.3. Results after Processing

The Data Processing helped in reducing the total count of characters considerably from 781989 to 500562 and also the total unique characters also reduced from 95 to 87. This reduced the number of batches and inturn reduced the computing time considerably.The generated tunes are more of music now rather than metadata but accuracy of the model did not increase much. The tunes generated are better post processing the data and the computation time is less but with same accuracy. Putting aside the accuracy the music generated by the model is smooth although not of professional quality, but it is smooth without any distortions and disturbances. I wanted to explore more and try to increase the accuracy and try to check the music generated with better accuracy.

5.4. Variations

I wanted to try different variations and try modifying hyperparameters to get a better result and more smooth music. I tried modifying the optimizer and tweaking with different parameters but the accuracy still dropped . Hence restoring the previous best model I have tried increasing the Dropout value from 0. 2 to 0. 4 for each of the LSTM layers and found that accuracy dropped suddenly. This gave me an intuition that model might not be overfitting and hence I have removed the dropouts from the model and the accuracy started increasing. I have implemented the same model to both the initial data and the preprocessed data and compared the accuracy in all different scenarios. The details in the below Fig.8 clearly shows the comparison.

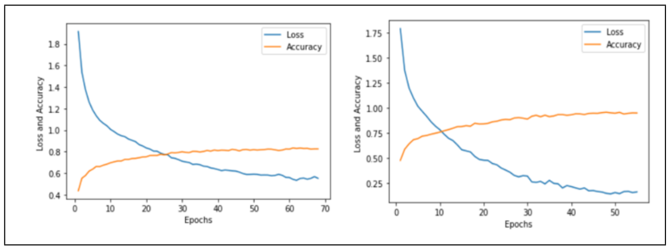


Figure 9. The left figure is with dropout and right one is without dropout.The model without dropout is overfitting as seen from the graph.

The variation of accuracy and loss with each epoch can be seen by plotting loss and accuracy with epochs. The variation with dropout and without dropout can also be seen in Fig.9. The model is overfitting as clearly seen from Fig.9, its accuracy is too high and loss is near to zero which clearly shows overfitting.

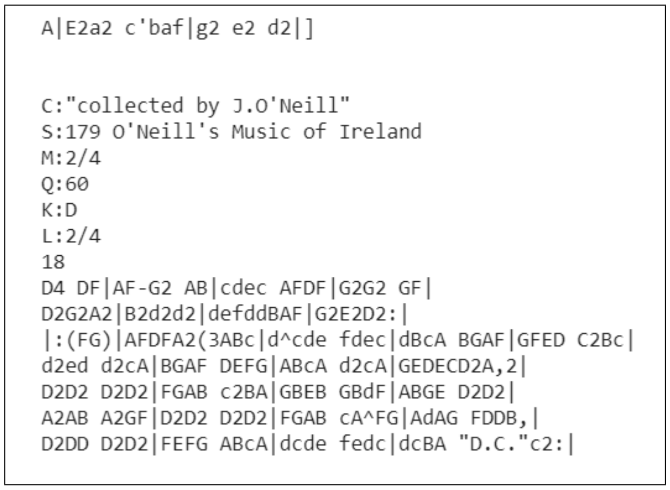
5.5. OverFitting

It can be seen from the image that without dropouts the model is overfitting. Overfitting in this case can be inferred as model trying to learn the input data completely and when trying to generate music it will recreate similar music. But we want our model to generate tunes which are smooth but are different from the original tunes. So our model should try to predict smooth music but not replicate the input music hence we should use dropouts to reduce ovverfitting of our model. We should have a trade-off between accuracy and originality of music such that the music generated should be able to generate smooth music and not overfit.

5.6. Output

While generating output the model asks user to input the initial character and the length of the music sequence user wants the model to generate. The model then takes the initial character and starts generating the next characters. The model will generate characters till the length of the generated music is same as the length given be the user. The model can start with a completely random character given by the user and starts generating next characters continuously until it reaches new lines which in our training data represents as ending of a tune. Post the new line the characters, model starts generating the music rhythm which will be of similar structure to abc tune. This can be clearly seen in the Fig.10 where the initial line of the output is not in a proper format.Post the new lines the model starts to generate characters resembling abc format.

To remove such lines I am ignoring the starting string of a generated sequence which helps in removing such unnecessary out of sync tunes.The correct sequence it start generating from next line onwards which I am consider-



```
A|E2a2 c'baf|g2 e2 d2|]


C:"collected by J.O'Neill"
S:179 O'Neill's Music of Ireland
M:2/4
Q:60
K:D
L:2/4
18
D4 DF|AF-G2 AB|cdec AFDF|G2G2 GF|
D2G2A2|B2d2d2|defddBAF|G2E2D2:|
|:(FG)|AFDFA2(3ABc|d^cde fdec|dBcA BGAF|GFED C2Bc|
d2ed d2cA|BGAF DEFG|ABcA d2cA|GEDECD2A,2|
D2D2 D2D2|FGAB c2BA|GBEB GBdF|ABGE D2D2|
A2AB A2GF|D2D2 D2D2|FGAB cA^FG|AdAG FDDB,|
D2DD D2D2|FEFG ABcA|dcde fedc|dcBA "D.C."c2:|
```

Figure 10. In fig is a tune generated by the trained model.The input given is to start the output with character A. The first line of generated tunes is not in a proper abc format. Post the two new lines the tune generated is in a proper abc format

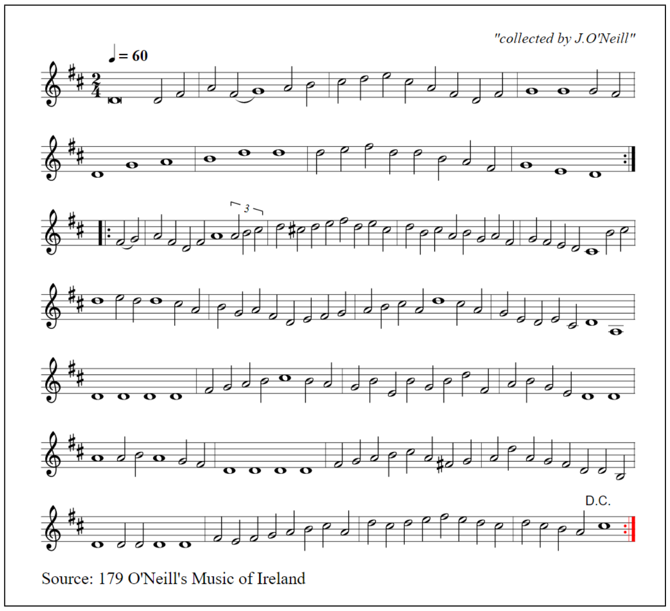


Figure 11. Sheet Music format of the above Generated text

ing.This helps the tune generated to start in a professional way, which is a common hurdle in music generation.

6. Conclusion and Future Works

6.1. Conclusion

I have used char-based RNN with LSTM cells on abc format music files. I have trained the model and achieved accuracies varying from 75 to 95.From my observations and findings I want to conclude that with an accuracy of 80-85% we can generate new music by learning patterns between musical notes. The music generated with this accuracy is fairly original and are as melodious as the original tunes without any breaks or discontinuities. There are limitations such as tunes and availability of tunes in abc format because of which the tunes are not of professional quality, if we can get abc formats of some of the famous and latest tunes we can make better contemporary music. I tried implementing the project original way without any libraries/toolkits but currently there are python toolkits like Music21 which can be of great help.

6.2. Future Works

There is still scope for improvements and research in the field of music generation. I have used only one type of music as training data and generated similar kind of music, we can try training different instruments and generate different tunes. We can also try training a complete song with all music instruments combined and train model such that it generated professional songs. With more knowledge on music formats and different kinds of starting tones of music we can limit the model to take only specific characters as inputs rather than taking all unique characters as inputs. We can also try different data augmentation in case the dataset is small and try generating music. There is scope for more research, to find out the best accuracy for a model such that it is not overfitting and generating fresh tunes.

References

[1] O'neill's music of ireland, mit.edu. 1

[2] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory, LSTM. 9(8), 1997. 1

[3] Alex Issa. Unrolled rnn, ImgRef. 2019. 2

[4] West Jeremy; Ventura Dan; Warnick Sean. Transfer_learning,TL. 2007. 4

[5] Gaurav Sharma. Music generation using deep learning,ImgRef. 2018. 4

[6] Gaurav Sharma. Music generation using deep learning,ImgRef. 2018. 3

[7] Gaurav Sharma. Music generation using deep learning,ImgRef. 2018. 2

[8] Gaurav Sharma. Music generation using deep learning,ImgRef. 2018. 2

[9] Sigurur Skúli. How to generate music using a lstm neural network in keras, tds, 2017. 1

[10] Chris Walshaw. Abc notation, ABC. 2008. 1