

Classical Music Generation using RNN

Project report submitted in partial fulfillment of the requirement for
the degree of Bachelor of Technology
in
Computer Science and Engineering

By

Priyanshu Sharma (181234)

Under the supervision of

Dr. Rajni Mohana

to

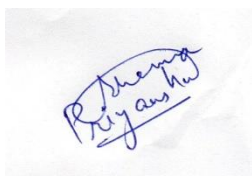


Department of Computer Science & Engineering and Information
Technology
**Jaypee University of Information Technology Waknaghat, Solan-
173234, Himachal Pradesh**

Candidate's Declaration

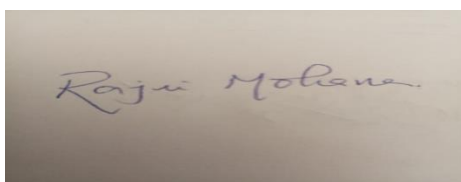
I hereby declare that the work presented in this report entitled “**Classical Music Generation using Recurrent Neural Network**” in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science and Engineering/Information Technology** submitted in the department of Computer Science & Engineering and Information Technology, Jaypee University of Information Technology Warknaghat is an authentic record of my own work carried out over a period from January 2022 to May 2022 under the supervision of **Dr. Rajni Mohana** (Associate Prof. in CSE department).

The matter embodied in the report has not been submitted for the award of any other degree or diploma.

A handwritten signature in blue ink that reads "Priyanshu Sharma".

Priyanshu Sharma, 181234

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

A handwritten signature in black ink that reads "Rajni Mohana".

Dr. Rajni Mohana
Associate Professor
Computer Science
Dated: 04-12-2021

AKCNOWLEDGEMENT

Firstly, I express my heartiest thanks and gratefulness to almighty God for His divine blessing makes us possible to complete the project work successfully.

I am really grateful and wish my profound indebtedness to Supervisor Dr. Rajni Mohana, Associate Professor, Department of CSE Jaypee University of Information Technology, Wakhnaghat. Deep Knowledge & keen interest of my supervisor in the field of “Application Development” to carry out this project. Her endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stage have made it possible to complete this project.

I would like to express my heartiest gratitude to Dr. Rajni Mohana, Department of CSE, for his kind help to finish my project.

I would also generously welcome each one of those individuals who have helped me straight forwardly or in a roundabout way in making this project a win. In this unique situation, I might want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated my undertaking.

Finally, I must acknowledge with due respect the constant support and patients of my parents.

Priyanshu Sharma
(181234)

TABLE OF CONTENT

S.NO.	TITLE	PAGE NO.
-	List of abbreviations	IV
-	List of figures	V-VI
-	ABSTRACT	VII
1	INTRODUCTION	-
1.1.1	Concept of Music	1-2
1.1.2	Machine Learning	2-5
1.1.3	Categories of machine learning	5-7
1.1.4	Deep Learning	7-8
1.1.5	Introduction to LSTM	9-11
1.2	PROBLEM STATEMENT	11
1.3	OBJECTIVES	12
1.4	METHODOLOGY	12
2	LITERATURE REVIEW	-
2.1	Research Article	13
2.2	Research Article	13-15
2.3	Article	15-16
2.4	Blog	16
2.5	Blog	16
3	SYSTEM DEVELOPMENT	-
3.1	Feedforward Neural Network	17-20
3.2	Recurrent Neural Network	20-32
4	PERFORMANCE ANALYSIS	33-41
5	CONCLUSION	42
-	REFERENCES	43
-	APPENDIX	44-47

LIST OF ABBREVIATIONS

Abbreviation	Full Form
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
GRU	Gated Recurrent Unit
XML	Extensible Markup Language
AI	Artificial Intelligence
ML	Machine Learning
DL	Deep Learning

LIST OF FIGURES

Figure No.	Figure Title	Page No.
1.1	AI vs ML vs DL	7
1.2	Typical Neural Network	8
1.3	Loading important dependencies	9
1.4	Loading text and making character-to-integer mapping	9
1.5	Preparing dataset	10
1.6	Reshaping of x	10
1.7	Defining LSTM model	10
1.8	Fitting model and generating characters	11
1.9	Output	11
2.1	Determining repetition stride as no. of steps in RNN	14
2.2	GRU vs LSTM results	15
3.1	Overview of Structure of LSTM	17
3.2	A sigmoid function	18
3.3	Visualization of a node	18
3.4	Visualization of Layer	19
3.5	Visualization of a neural network	19
3.6	Visualization of layers as single objects	19
3.7	Feeding output of node to itself	22
3.8	Unwrapping the network along time axis	23
3.9	Visualization of transferring memory cell data alongside output	24
3.10	Working of Convolution	26
3.11	System invariant in both time and notes	27
3.12	Having patterns in time and note space without loss in invariance	28
3.13	Considering time connections as loops	29
4.1	Data Processing Overview	33

4.2	Overview of model	34
4.3	Probability of each class	35
4.4	Adaptive Moment Estimation used	36
4.5	Weights of trained model loaded to new model	36
4.6	Black-box like model of the project flow	37
4.7	No of midi files	37
4.8	Counting no. of nodes	38
4.9	Counting no. of pitches	38
4.10	Input and output of data of model	39
4.11	Model Training	40
4.12	Musical Notes as Output	40
4.13	Model Summary	41

ABSTRACT

We want to train a neural network to compose music. To be precise, we would want the network to produce euphonic music composed of chorus and variation in notes, thus resulting in classical music. We would like the network to consider periods of repeated melodies and overlapping timesteps like chorus, to create music progressively based on the existent melodies in the previous timesteps. We will have to break the structure of network down to each time step as we want the network to choose indefinitely. To achieve this, each node of the network has to evaluate information from the previous nodes and give its learnings to the next node. This process that we described above will be performed using Recurrent Neural Networks (RNN).

A recurrent neural network is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. In the basic feedforward network, there is a single direction in which the information flows, i.e., from input to output, but in a recurrent neural network, this direction constraint does not exist. We will be working with a simple form of such RNN. The power of this is that it enables the network to have a simple version of memory, with very minimal overhead. However, this implementation is not complete yet. The main problem with is that the memory is very short-term. Any value that is output in one time step becomes input in the next, but unless that same value is output again, it is lost at the next tick. For solving this problem, we will be making use of Long Short-Term Memory (LSTM) node instead of a normal node. This introduces a memory cell value that is passed down for multiple time steps, and which can be added to or subtracted from at each tick.

CHAPTER -1 INTRODUCTION

INTRODUCTION

Concept of music

Music can be conceived of as a collection of distinct tones with different frequencies. The technological definition will be determined by the musical instrument utilised in the context. For instance, music is actually composed of little parts known as notes and songs in the case of a pianist. Note the sound produced by one key, whereas the song is the sound produced by two or more keys at the same time. Typically, most songs have three or more keys at any given time. The octave pattern is the repetitive pattern. When words alone aren't enough, music can be thought of as a language created to convey a person's emotions. It has a few elements that we'd like to list here: tone, melody, harmony, rhythm, texture, timbre, speech, and mood. So, since the lyrics must be similar, the challenge is to find the difference between programme design and musical production. Fortunately, there is a music commentary system and digital partitions to encrypt it in plain text file formats that can be easily differentiated and used. The B music file note is an example of such a format.

As a result, default music generation can be thought of as a process that produces a small or short piece of music that will need to say no in order to intervene a little from the person. There are several ways to accomplish this, two of which are to utilise wavenet or to use LSTM. Wavenet is a production model developed by Google DeepMind that is based on in-depth reading of crude audio. The wavenet is known as a productive model since the main purpose of the network is to produce new samples from the actual distribution of data. The second method employs LSTM, an RNN variant capable of capturing long-term dependence on the input sequence. LSTM is popular today since it is utilised in word-for-word comparisons, video segmentation, text summaries, and so on. We will use the second method, LSTM, for this project. The rude sound wave episode, which means the representation of the wave in the domain of the timeline, is provided as included. The timeline series allows us to represent the

sound wave as a succession of magnitude waves recorded at different moments. We attempted to predict the successive amplitude value after obtaining an input such as the amplitude wave sequence. In each step of our LSTM example, the amplitude value is entered into the LSTM, which then encases the hidden vector and transmits it in the next step. The current hidden vector $h(t)$ is calculated based on the current input $a(t)$ and the previously hidden vector $h(t-1)$. This is a common procedure for how sequential information is captured on any RNN. Using LSTM to capture input sequence information is a great advantage, but it also takes a long time due to the input processing sequence.

There are metrics and ways to support that there are ways to check the quality of music. We can define metrics such as note repetition, non-key note notes, motif note notes, repetitive motif notes, fixed leaps scale, initial composition with tonic ration, and so on, based on the rules of musical theory. From a mathematical perspective, it is possible to assess the quality of the music produced by calculating the similarities between the music produced and the music used to train the model.

Machine Learning

Machine learning is an artificial intelligence (AI) program that gives systems the ability to automatically learn and develop from experience without being clearly planned. Machine learning focuses on the development of computer programs that can access data and use it for self-study. The learning process begins with observations or data, for example, specific information, or instructions, so that we look at patterns in the data and make better decisions in the future based on the examples we provide. The main purpose is to allow computers to read automatically without human intervention or help and to correct actions appropriately. However, using old machine learning algorithms, text is considered a sequence of keywords; rather, a semantic-based approach mimics a person's ability to understand the meaning of a text.

We know humans learn from their past experiences and machines follow instructions given by humans but what if humans can train the machines to learn

from the past data and do what humans can do and much faster well that's called machine learning. But it's a lot more than just learning, it is also about understanding and reasoning. So we will learn about the basics of machine learning. Consider a boy Paul who loves listening to new songs. He either likes them or dislikes them. Paul decides this on the basis of the song's tempo, genre, intensity and the gender of voice. For simplicity let's just use tempo and intensity for now. So here tempo is on the x axis ranging from relaxed to fast whereas intensity is on the y axis ranging from light to soaring. We see that Paul likes the song with fast tempo and soaring intensity while he dislikes the song with relaxed tempo and light intensity. So now we know Paul's choices. Let's say Paul listens to a new song. A song a has fast tempo and a soaring intensity so it lies somewhere. Looking at the data can you guess whether Paul will like the song or not correct so Paul likes this song. By looking at Paul's past choices we were able to classify the unknown song very easily. Let's say now Paul listens to a new song let's label it as song 'b' so song 'b' lies somewhere here with medium tempo and medium intensity neither relaxed nor fast neither light nor soaring. Now can you guess whether Paul likes it or not. Not able to guess whether Paul will like it or dislike it are the choices unclear. We could easily classify song a but when the choice became complicated as in the case of song 'b' and that's where machine learning comes in. Let's see how in the same example for song 'b' if we draw a circle around the song b we see that there are four votes for like whereas one would for dislike if we go for the majority votes we can say that Paul will definitely like the song that's all this was a basic machine learning algorithm also it's called k-nearest neighbors. So this is just a small example in one of the many machine learning algorithms. Such algorithm is easy but what happens when the choices become complicated as in the case of song 'b'. That's when machine learning comes in. It learns the data, builds the prediction model and when the new data point comes in, it can easily predict for it. More the data better the model, and higher will be the accuracy. There are many ways in which the machine learns. It could be either supervised learning unsupervised learning or reinforcement learning. Let's first quickly understand supervised learning.

Suppose your friend gives you one million coins of three different currencies. Say one rupee, one euro and one dirham. Each coin has different weights, for example, a coin of one rupee weighs three grams, one euro weighs seven grams and one dirham weighs four grams. Your model will predict the currency of the coin. Here your weight becomes the feature of coins while currency becomes the label. When you feed this data to the machine learning model it learns which feature is associated with which label. For example, it will learn that if a coin is of 3 grams, it will be a 1-rupee coin let's give a new coin to the machine. On the basis of the weight of the new coin, your model will predict the currency. Hence supervised learning uses labeled data to train the model as here the machine knew the features of the object and also the labels associated with those features. On this note let's move to unsupervised learning and see the difference. Suppose you have cricket data set of various players with their respective scores and wickets taken. When you feed this data set to the machine, the machine identifies the pattern of player performance, so it plots this data with the respective wickets on the x-axis while runs on the y-axis. While looking at the data you will clearly see that there are two clusters. The one cluster are the players who scored higher runs and took less wickets while the other cluster is of the players who scored less runs but took many wickets. So here we interpret these two clusters as batsmen and bowlers. The important point to note here is that there were no labels of batsmen and bowlers. Hence the learning with unlabeled data is unsupervised learning. So we saw supervised learning where the data was labeled and the unsupervised learning where the data was unlabeled. Then there is reinforcement learning which is a reward based learning or we can say that it works on the principle of feedback. Here let's say you provide the system with an image of a dog and ask it to identify it. The system identifies it as a cat so you give a negative feedback to the machine saying that it's a dog's image. The machine will learn from the feedback and finally if it comes across any other image of a dog it will be able to classify it correctly. This is reinforcement learning. Input is given to a machine learning model which then gives the output according to the algorithm applied. If it's right we take the output as a final result else we provide feedback

to the training model and ask it to predict until it learns. I hope you've understood supervised and unsupervised learning. So let's test ourselves on determining whether the given scenario uses supervised or unsupervised learning. Scenario one, facebook recognizes your friend in a picture from an album of tagged photographs. Scenario two, netflix recommends new movies based on someone's past movie choices. Scenario three, analyzing bank data for suspicious transactions and flagging the fraud transactions. Don't you sometimes wonder how is machine learning possible in today's era. Well that's because today we have humongous data available. Everybody is online either making a transaction or just surfing the internet and that's generating a huge amount of data every minute and that data is the key to analysis. Also the memory handling capabilities of computers have largely increased which helps them to process such huge amount of data at hand without any delay. And yes computers now have great computational powers. So there are a lot of applications of machine learning out there. To name a few, machine learning is used in healthcare where diagnostics are predicted for doctor's review. The sentiment analysis that the tech giants are doing on social media is another interesting application of machine learning. Fraud detection in the finance sector and also to predict customer churn in the e-commerce sector while booking a cab, you must have encountered surge pricing often where it says the fare of your trip has been updated. Well that's an interesting machine learning model which is used by global taxi giant uber and others where they have differential pricing in real time based on demand, the number of cars available, bad weather, rush etc. So they use the search pricing model to ensure that those who need a cab can get one. Also it uses predictive modeling to predict where the demand will be high with the goal that drivers can take care of the demand and search pricing can be minimized.

Categories of Machine Learning

Machine learning algorithms are generally classified as either supervised or unsupervised: Supported machine learning algorithms can apply past lessons to new data using labelled examples to predict future events. The learning algorithm

generates targeted activity to make predictions about output values based on the analysis of a well-known training database. After adequate training, the system is capable of providing the objectives of any new installation. The learning algorithm can also compare the output to the correct, targeted output and detect errors in order to modify the model appropriately.

Unsupervised machine learning algorithms, on the other hand, are used when the information used for training may be segregated and labelled. Unrestricted learning lessons on how systems can handle the task of defining a hidden structure from unlabeled data. The system does not detect the incorrect output, but it scans the data and may draw clues from the data sets to describe hidden properties in unlabeled data.

Semi-supervised machine learning algorithms fall between supervised and unsupervised learning because they train with both labelled and unlabeled data - often a small amount of labelled data and a large amount of unlabeled data. Systems that employ this method are capable of significantly improving learning accuracy. Supervised learning is typically preferred when the labelled data received necessitates the use of consistent and appropriate resources to train / learn from it. If not, obtaining data without labels usually does not necessitate additional resources.

Reinforcement machine learning algorithms to boost the learning process that interacts with its environment by producing actions and detecting errors or rewards. The most important aspects of reinforcement learning are test and error search and delayed rewards. This approach enables machines and software agents to automatically determine good behaviour in a certain context in order to maximise their effectiveness. A simple reward answer is required for the agent to determine which action is the best; this is known as a confirmation signal.

Supervised Learning		Unsupervised Learning
Classification	Regression	Clustering
Support Vector Machines	Linear Regression	K-Means
Discriminant Analysis	SVR	Fuzzy C-Means
Naive Bayes	Decision Trees	Hidden Markov Model
Nearest Neighbor	Neural Networks	Hebbian Learning

Table 1.1. Supervised vs Unsupervised Learning

Deep Learning

Deep learning can be thought of as a subset of machine learning, which, in turn, can be thought of as a subset of Artificial intelligence, as we've discussed previously. Artificial Intelligence is a broad term that refers to techniques that enable computers to mimic human behavior. Machine Learning represents a set of algorithms trained on data that make all of that possible.

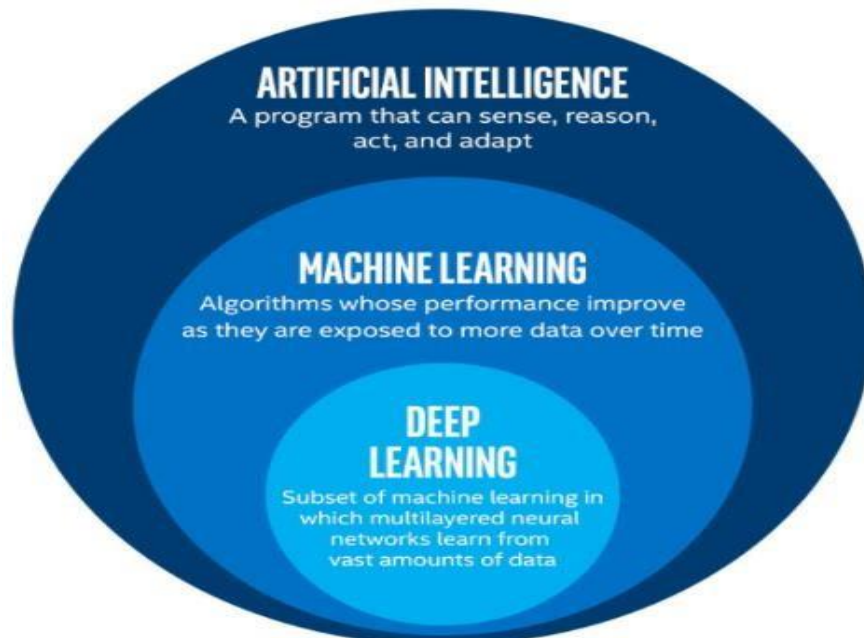


Fig 1.1. AI vs ML s DL

Deep Learning, on the other hand, is just a type of Machine Learning, inspired by the structure of a human brain. Deep learning algorithms attempt to draw similar conclusions as humans would by continually analyzing data with a given logical structure. To achieve this, deep learning uses a multi-layered structure of algorithms called neural networks.

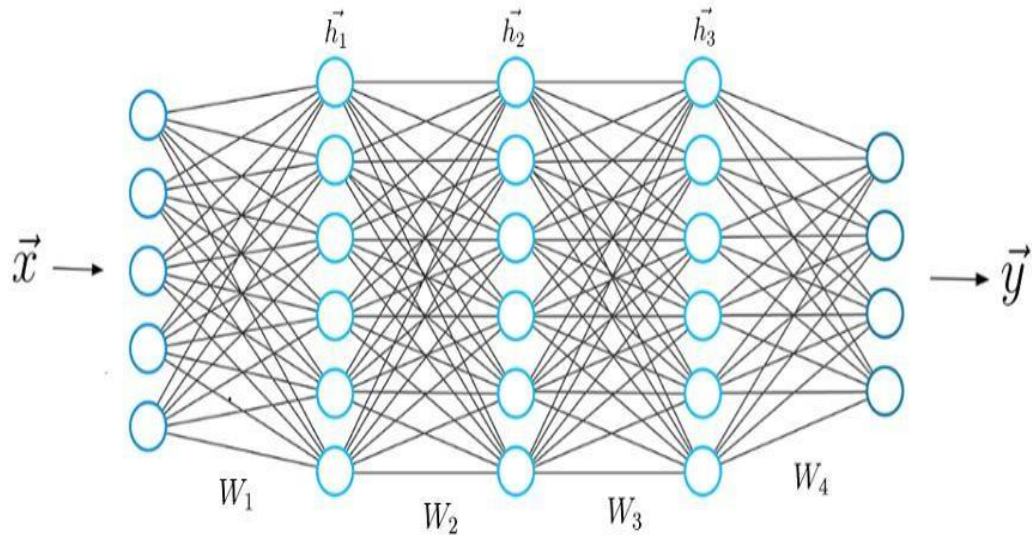


Fig1.2. Typical neural network

The neural network's structure is modelled after that of the human brain. Neural networks can be taught to do similar duties in data to how our brains spot patterns and separate different sorts of information.

Individual layers of neural networks can be viewed as a form of filter that functions from negative to concealed, increasing the likelihood of finding and creating a positive result. The human brain functions similarly. When we get new knowledge, our brain attempts to compare it to what we already know. Deep neural networks employ the same concept.

We may use neural networks to execute a variety of tasks, such as merging, splitting, and retrieving data. We can use neural networks to collect and arrange non-labeled data based on the similarities between the samples. In the instance of segregation, we can use a labelled database to train the network to divide the samples into different categories.

Introduction about LSTM

Problems with sequence prediction have been around for a long time. They are considered to be one of the most difficult problems in the data science business to solve. These include a variety of issues, such as sales forecasts and discovering patterns in stock market data, understanding movie episodes and determining your speaking style, and language translation and predicting your next name on your iPhone. With recent advances in data science, it has been discovered that in almost all of these sequence prediction problems, Short-Term Memory Networks (LSTMs) have been recognised as the most effective solution.

In many ways, LSTMs are limited over the neural networks of feed forwarding and RNN. This is because they choose long patterns to show their communal property. The goal of this article is to define LSTM and show you how to apply it to real-world problems.

Basic example of text generation using LSTMs for basic understanding how they work:

```
# Importing dependencies numpy and keras
import numpy
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import LSTM
from keras.utils import np_utils
```

Fig1.3. Loading important dependencies

```
# Load text
filename = "/macbeth.txt"

text = (open(filename).read()).lower()

# mapping characters with integers
unique_chars = sorted(list(set(text)))

char_to_int = {}
int_to_char = {}

for i, c in enumerate(unique_chars):
    char_to_int.update({c: i})
    int_to_char.update({i: c})
```

Fig1.4. Loading text and making character-to-integer mapping

```

# preparing input and output dataset
X = []
Y = []

for i in range(0, Len(text) - 50, 1):
    sequence = text[i:i + 50]
    Label =text[i + 50]
    X.append([char_to_int[char] for char in sequence])
    Y.append(char_to_int[Label])

```

Fig1.5. Preparing dataset

```

# reshaping, normalizing and one hot encoding
X_modified = numpy.reshape(X, (Len(X), 50, 1))
X_modified = X_modified / float(Len(unique_chars))
Y_modified = np_utils.to_categorical(Y)

```

Fig1.6. Reshaping of x

```

# defining the LSTM model
model = Sequential()
model.add(LSTM(300, input_shape=(X_modified.shape[1], X_modified.shape[2]), return_sequences=True))
model.add(Dropout(0.2))
model.add(LSTM(300))
model.add(Dropout(0.2))
model.add(Dense(Y_modified.shape[1], activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam')

```

Fig1.7. Defining LSTM model

```

# fitting the model
model.fit(X_modified, Y_modified, epochs=1, batch_size=30)

# picking a random seed
start_index = numpy.random.randint(0, Len(X)-1)
new_string = X[start_index]

# generating characters
for i in range(50):
    x = numpy.reshape(new_string, (1, Len(new_string), 1))
    x = x / float(Len(unique_chars))

    #predicting
    pred_index = numpy.argmax(model.predict(x, verbose=0))
    char_out = int_to_char[pred_index]
    seq_in = [int_to_char[value] for value in new_string]
    print(char_out)

    new_string.append(pred_index)
    new_string = new_string[1:Len(new_string)]

```

Fig1.8. Fitting model and generating characters

```

a
c
b
.
t
h
e
t
o
e

```

Fig1.9. Output

PROBLEM STATEMENT

Develop a model which successfully generates classical music. The model is designed as such that it takes a small sample of music as input and generates a bigger music sample by itself with minimal to no human intervention.

OBJECTIVES

At the end of completing this project, we want the model to successfully generate music samples by taking a short sample of notes as input. The model should not just work with certain kind of inputs but should rather work with any kind of music.

METHODOLOGY

A chunk of raw audio wave is given as input, and it refers to the representation of a wave in the time series domain. Time series domain allows us to present an audio wave in the form of magnitude waves that are recorded at different points in time. We try to predict the successive amplitude value after acquiring the input as a sequence of amplitude waves. In our LSTM case, an amplitude value is supplied into the LSTM, which then computes the hidden vector and passes it on to the next time steps. The current hidden vector $h(t)$ is computed using the current input $a(t)$ and the previously hidden vector $h(t-1)$. This is just the standard procedure for capturing sequential information in any RNN.

CHAPTER -2 LITERATURE SURVEY

RESEARCH ARTICLE

Alexander Agung Santoso Gunawan, Ananda Phan Iman, Derwin Suhartono, “Automatic Music Generator Using Recurrent Neural Network”, International Journal of Computational Intelligence Systems Vol. 13(1), 2020, Published by Atlantis Press SARL, pp. 645-654

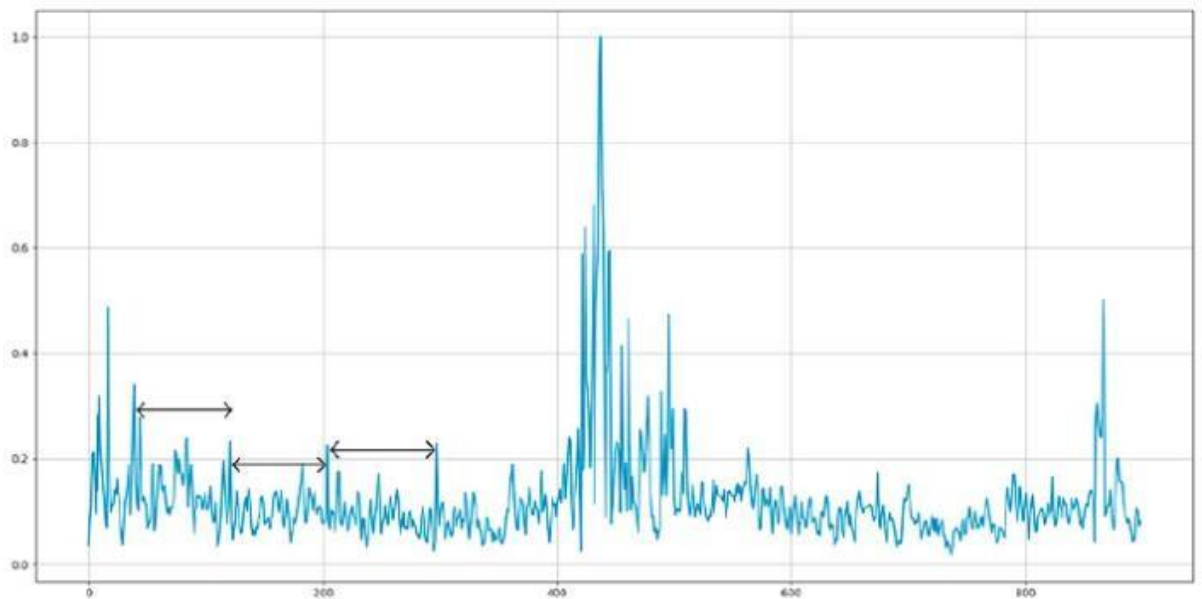
When the authors used midi as an input file in this work, they constructed a default music generator. The generator and test model were built using the LSTM network and GRUs. Midi collection, midi coding, music generating model training, music production, music testing, and midi recording are all part of their approach. They began by converting the midi file to a midi matrix using the midi encoding procedure. Then, as a production model, each component is trained in a single layer and a double layer model for each network. After that, they trained the LSTM and GRU-based classification model and used it as the objective test to evaluate the performance of each generator model that categorises each midi by musical era. Indeed, the authors spoke with persons with a musical background, such as those who have worked with or are interested in classical music, composers, performers, and digital composers. Their findings reveal that the double GRU model performs better, as it has a 70% similarity to a composer pattern in music. Furthermore, independent testing reveals that the music created is well received, scoring 6.85 out of 10 on a GRU of a double layer.

RESEARCH ARTICLE

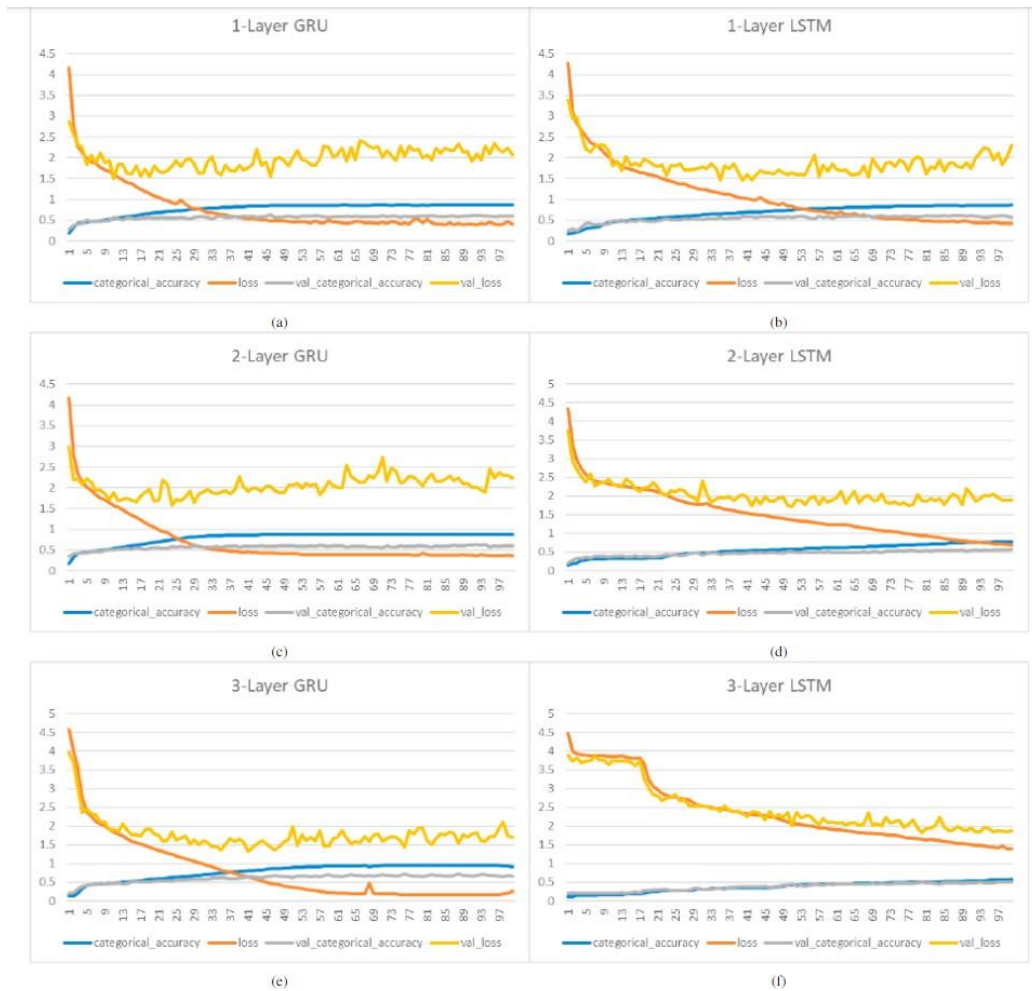
Alexandru-Ion Marinescu, “generating classical music using recurrent neural networks”, 23rd International Conference on Knowledge-Based and Intelligent Information & Engineering Systems, 2019,). Published by Elsevier B.V.

The author has worked on the integration of classical music in the form of music schools by providing an analysis of various RNN structures. The author discussed side-by-side comparisons of two of the most extensively used neural network layers, LSTM and GRU, respectively. You've looked at the effects of changing architecture metaparameters like the number of hidden neurons, the layer calculation, and the number

of epochs in terms of phase accuracy and losses. He also undertook model work on other pieces of music, which is a method of measuring repetition in a piece of music. This is done because it is regarded as a crucial factor in deciding the length of inputs to be maintained during training. They initially believed that music was equivalent to words since, in the author's opinion, music is a form of language. As a result, the goal was to create a balance between programme creation and musical production. The author then discusses the musical commentary system and its digital counterparts, which are simple to process and use. Popular ABC notation and XML notation are two examples of such notes. The main method was also used, and they did so in the language python system, using the Keras and Tensorflow framework as a backend. The number of GRU layers employed was three, which were stacked on top of each other, and he used a 512 hidden neuron number. Later, the same procedure was used with LSTM layers and the computation of the hidden neuron and number of layers to determine which method was better for the job. The authors used an empirical method to determine the number of RNN steps by checking the repetition stride.



Graph2.1. Determining repetition stride as no. of steps in RNN



Graph2.2. GRU vs LSTM results

ARTICLE

Qibin Lou, “Music Generation using Neural Networks”, Stanford University

To explore numerous styles and discover the most appropriate way, the author has worked on the construction of music employing a variety of methods. Melody RNN, a Google open source project called as Magenta, was referenced by the author. Following that, the author discusses Biaxial RNN. Following that, the author discusses wavenet, another model we discussed briefly earlier. Finally, the author explains the findings, discovering that if given the main reference, the Melody-RNN model produces a piece of piano music that is actually audible, albeit monophonic. When discussing Biaxial-

RNN, on the other hand, the author discovers that this model provides an outstanding musical composition with rhythm.

He stated that he was looking forward to the many future projects that will be undertaken in this project. This includes acquiring additional training data and transferring the project to a specific cloud platform that will help implement and test code faster as doing so on your system will take a lot of time as training data on your system takes a lot of time.

WEBSITE BLOG

<https://www.danieldjohnson.com/2015/08/03/composing-music-with-recurrent-neural-networks/>

This website blog was very interesting to research as the author spoke very well about the various aspects of the whole process of producing music by RNNs while very brief and following the route to the destination. The author begins by talking about how he perceives the concept of music and then goes on to relate this to feedforward neural networks. The author continues to speak about how he perceives RNNs and how he believes the problem can be linked to or solved by RNNs. He then went on to speak about how the nervous system may be trained. The author has made it clear that the result of any input can be used to assess whether something is good or bad. The author then considers if all of this can, in the end, produce music. And the author continues to speak about it, eventually coming to the conclusion that we can employ Biaxial RNNs to achieve the ultimate goal of producing music.

WEBSITE BLOG

<https://towardsdatascience.com/music-generation-through-deep-neural-networks-21d7bd81496e>

This blog begins the conversation by talking about representing the music of machine learning models. The author spoke and used the ABC music commentary in his interview. The discussion goes on to discuss details about music databases, data processing, model selection, multi-RNNs, time-dense layer, mean layers, stops, softmax layer, optimizer, and finally music generation.

CHAPTER -3 SYSTEM DEVELOPMENT

In traditional machine learning models, we cannot store a model's previous stages. However, we can store previous stages with RNN.

A repeating module in an RNN takes input from the previous stage and outputs it as input to the following stage. RNNs, on the other hand, can only remember data from the most recent stage, therefore our network will require more memory to learn long-term dependencies. LSTMs come to the rescue in this situation.

LSTMs are a type of RNN that has a chain-like topology but a different repeating module structure than RNNs.

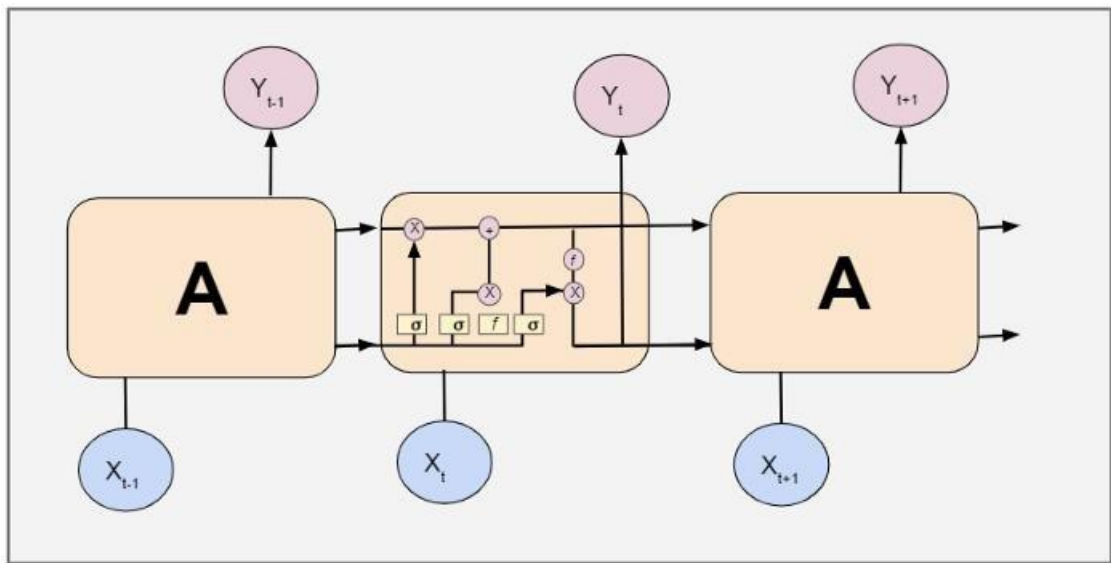


Fig3.1. Overview of Structure of LSTM

Feedforward Neural Networks:

In a basic neural network, one node takes a particular quantity of input, multiplies it by a specific weight, and then combines it all. The whole sum is then lowered to a range (typically -1 to 1 or 0 to 1) using an indirect opening function, such as a sigmoid function, by adding another fixed (called "bias").

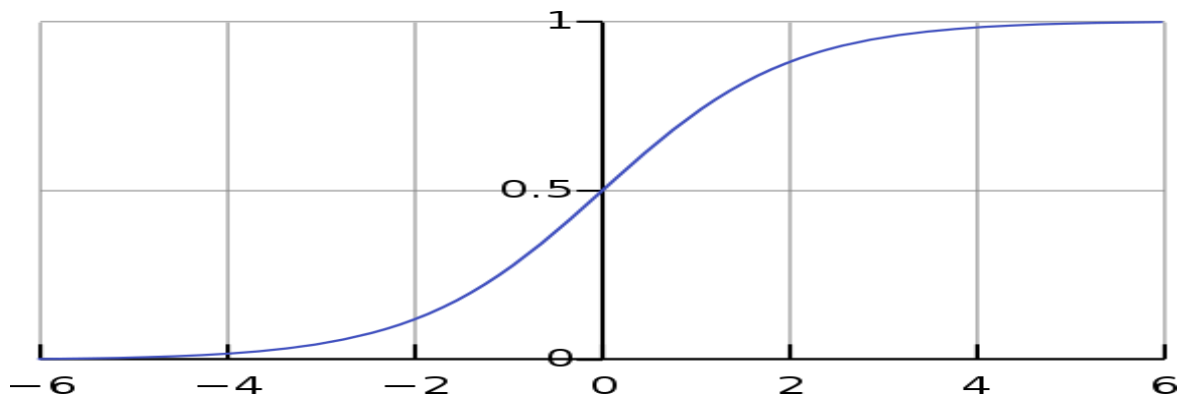


Fig3.2. A sigmoid function.

We can visualize this node by drawing its inputs and single output as arrows, and denoting the weighted sum and activation by a circle:

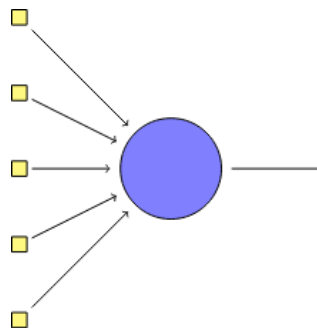


Fig3.3. Visualization of a node

We can then take multiple nodes and feed them all the same inputs, but allow them to have different weights and biases. This is known as a layer.

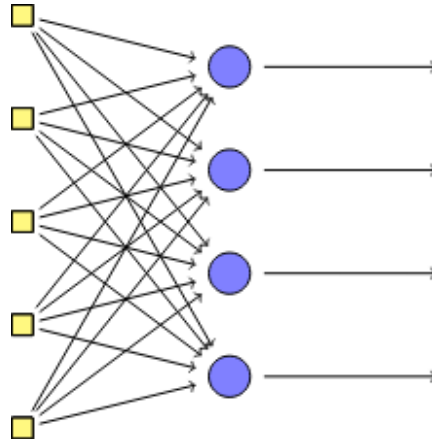


Fig3.4. Visualization of Layer

Because each node in the layer makes a weighted sum, but they all share the same input, we can calculate the output using matrix multiplication, followed by activating the element. This is one of the reasons why neural networks can be trained so effectively.

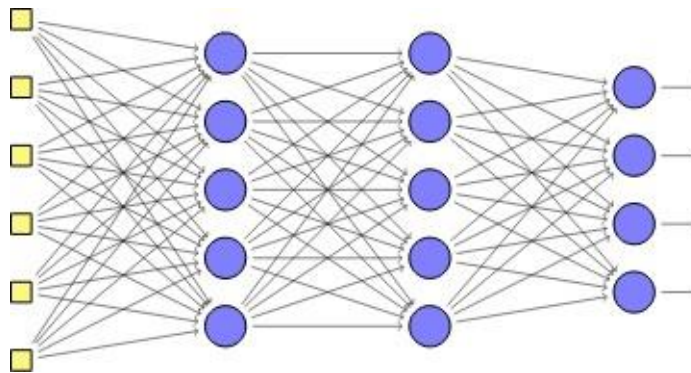


Fig3.5. Visualization of a neural network

Then we can connect multiple layers together and now, we have a neural network. The set of inputs is called the “input layer”, the last layer of nodes is called the “output layer”, and all intermediate node layers are called “hidden layers”. Since each node has a single output value, all the arrows from each node carry the same value.



Fig3.6. Visualization of layers as single objects

To make it easier, we can visualize layers as a single object, because that is how they are used most of the time, so, from this point on, you see a single circle, representing the entire network layer, and the arrows represent value vectors.

Recurrent Neural Networks

In a basic feedforward network, there is only one way in which information flows: from input to output. But in a progressive neural network, this directional conclusion does not exist. There are many networks that may not be categorized as duplicate, but we will focus on one simple and very effective one.

Applications of machine learning have gotten a lot of traction in the last few years. There's a couple of big categories that have had wins. One is identifying pictures, the equivalent of finding cats on the Internet and any problem that can be made to look like that, and the other is sequence to sequence translation this can be speech to text or one language to another. Most of the former are done with convolutional neural networks and most of the latter are done with recurrent neural networks. A particularly long short-term memory to give an example of how long short-term memory works. We will consider the question of what's for dinner. Let's say for a minute that you are a very lucky apartment dweller and you have a flatmate who loves to cook dinner every night. He cooks one of three things: sushi, waffles or pizza and you would like to be able to predict what you're going to have on a given night so you can plan the rest of your days eating accordingly. In order to predict what you're going to have for dinner, you set up a neural network. The inputs to this neural network are a bunch of items like the day of the week, the month of the year and whether or not your flatmate is in a late meeting. These are variables that might reasonably affect what you're going to have for dinner. You can think of them as a voting process and so in the neural network that you set up there's a complicated voting process and all of the inputs like day of the week and month of the year go into it, and then you train it on your history of what you've had for dinner. In this way, you learn how to predict what's going to be for dinner tonight. The trouble is that your network doesn't work very well. Despite carefully choosing your inputs and training it thoroughly, you still can't get much better than chance predictions on dinner

as is often the case with complicated machine learning problems. It's useful to take a step back and just look at the data and when you do that you notice a pattern your flatmate makes pizza, then sushi followed by waffles and then pizza again in a cycle. It doesn't depend on the day of the week or anything else. It's in a regular cycle so knowing this we can make a new neural network. In our new one the only inputs that matter are what we had for dinner yesterday. So if we know if we had pizza for dinner yesterday, then it will be sushi tonight and if sushi yesterday, then waffles tonight and if we had waffles yesterday, then pizza tonight. It becomes a very simple voting process and it's right all the time because your flatmate made it incredibly consistent. Now if you happen to be gone on a given night, let's say yesterday you were out, you don't know what was for dinner yesterday but you can still predict what's going to be for dinner tonight by thinking back to days ago. Think what was for dinner then so what would be predicted for you last night and then you can use that prediction in turn to make a prediction for tonight. So we make use of not only our actual information from yesterday but also what our prediction was yesterday. So at this point it's helpful to take a little detour and talk about vectors. A vector is just a fancy word for a list of numbers. If I want to describe the weather to you for a given day, I could say the high is 76 degrees Fahrenheit and the low is 43, the wind is 13 miles an hour, there's going to be a quarter inch of rain and the relative humidity is 83%. That's how the vector is the reason that it's useful. Numbers are computer's native language. If you want to get something into a format that it's natural for a computer to compute to do operations or to do statistical machine learning, then lists of numbers are the way to go. Everything gets reduced to a list of numbers before it goes through an algorithm. We can also have vector for statements like it's Tuesday. In order to encode this kind of information what we do is we make a list of all the possible values that could have. In this case, all the days of the week and we assign a number to each and then we go through and set them all equal to zero except for the one that is true. Right now this format is called one hot encoding and it's very common to see long vector of zeros with just one element being one. It seems inefficient but for a computer this is a lot easier way to ingest that information. So we can make a one hot vector for our prediction. For dinner tonight, we set everything equal to zero except for the dinner item that we predict. So in this case, we'll be predicting sushi. Now we can

group together our inputs and outputs into vectors which is separate lists of numbers and it becomes a useful shorthand for describing this neural network. So we can have our dinner yesterday vector and our prediction for today vector and the neural network is just connections between every element in each of those input vectors to every element in the output vector. And to complete our picture we can show how the prediction for today will get recycled. Now we can see how if we were lacking some information. Let's say we were out of town for two weeks, we can still make a good guess about what's going to be for dinner tonight. We just ignore the new information part and we can unwrap or unwind this vector in time, until we do have some information to base it on. Then just play it forward and when it's unwrapped it looks like this and we can go back as far as we need to and see what was for dinner and then just trace it forward. In this way, we can play out our menu over the last two weeks until we find out what's for dinner tonight. So this was just a nice simple example that showed recurrent neural networks.

Basically, what we do is take the output of each hidden layer, and feed it back to itself as an additional input. Each node of the hidden layer receives both a list of input from the previous layer and a list of the effects of the current layer in the last step. So if the input layer has 5 values, and the hidden layer has 3 nodes, each hidden node receives as input a total of $5 + 3 = 8$ values.

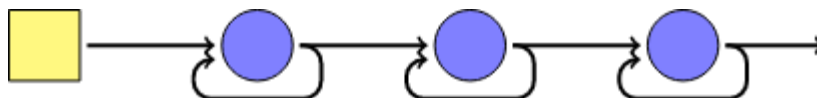


Fig3.7. Feeding output of node to itself

We can show this more clearly by unwrapping the network along the time axis:

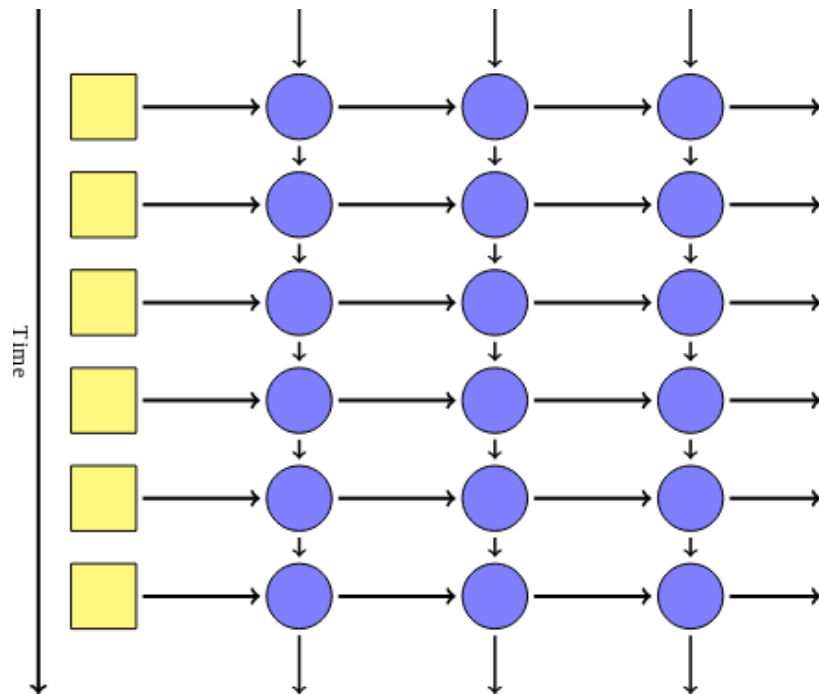


Fig3.8. Unwrapping the network along time axis

In this presentation, each horizontal row of layers is a network simultaneously. Each hidden layer receives both input from the previous layer and input from one previous step.

The strength of this is that it allows the network to have a simpler version of memory, with a much smaller overhead. This opens up the possibilities for input and output: we can supply input simultaneously, and allow the network to connect you using the status of each step.

One problem with this is that memory lasts a very short time. Any value that comes out of one step becomes the next input, but unless that value goes out again, it loses the next builder. To resolve this, we can use LSTM instead of the normal node. This introduces a number of “memory cells” that are transferred over a period of time, and can be added to or removed from each tick.

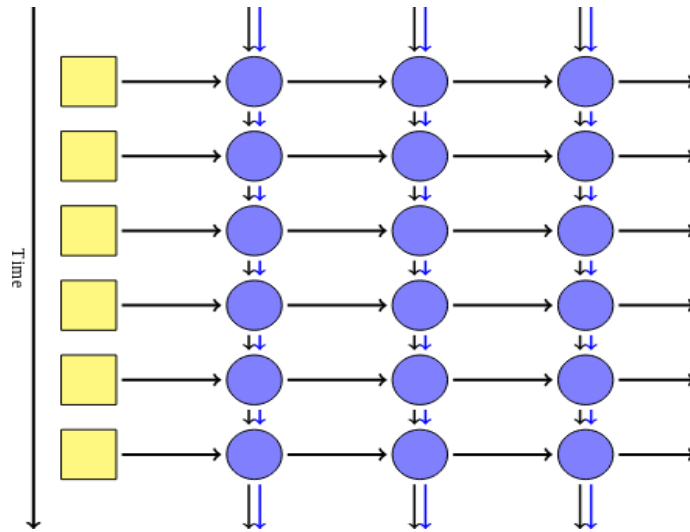


Fig3.9. Visualization of transferring memory cell data alongside output

The behavior of the neural network is determined by the weight set and bias each node has, so we need to adjust that to the right value.

First, we need to define what is good and what is bad, in terms of inclusion. This fee is called an expense. For example, if we were trying to use a neural network to model a mathematical activity, the cost may be the difference between the function response and the output of the network, which is square. Or if we were to try to make a model for the characters to appear in a certain order, the cost could be one to exclude the opportunity to predict the correct character each time.

Once we have this amount of costs, we can use backpropagation. This encompasses the calculation of cost trends in relation to weights (i.e. the derivative of the costs in relation to the weight of each node in each layer), and then using a specific method to improve weight correction to reduce costs. The bad news is that these changes are often very complex. But the good news is that many of them have already been done in libraries, so we can provide our gradient with the right service and allow it to adjust our weight accordingly.

Other usable enhancements include stochastic gradient downtime, Hessian-free performance, AdaGrad, and AdaDelta.

With our network design, there were a few structures we wanted to have:

- Have some understanding of time signature: I wanted to give the neural network its current time by referring to the time signature, as most music is associated with a fixed time signature.
- Consistency: I wanted the network to be able to create permanently, so it needed to be consistent at every step of the time.
- Be (often) consistent: Music can be freely changed up and down, and it stays that way. So, I wanted the neural network structure to be almost the same on each note.
- Allow multiple notes to be played at once, and allow the selection of matching songs.
- Allow the same note to be repeated: double C should be different than holding one C two bits.

Many RNN-based music creation methods do not change over time, as each step is a duplicate of a single network. But they usually do not change their comments. There is usually a specific output node that represents each note. So passing the whole thing up, let's say, one complete step, will produce a completely different result. In most cases, this is what you would like: "hello" completely different from "ifmmp", already "converted" into a single book. But in music, he wants to emphasize the related relationships over complete positions: large C songs sound like a larger D chord than a small C-chord, although a smaller C song is closer in relation to the positions of the full notes.

There is one type of widely used neural network today that has this flexible structure close to many indicators: convolutional neural networks for image recognition. This works by basically reading the convolution kernel and applying that convolution kernel to all pixel input images.

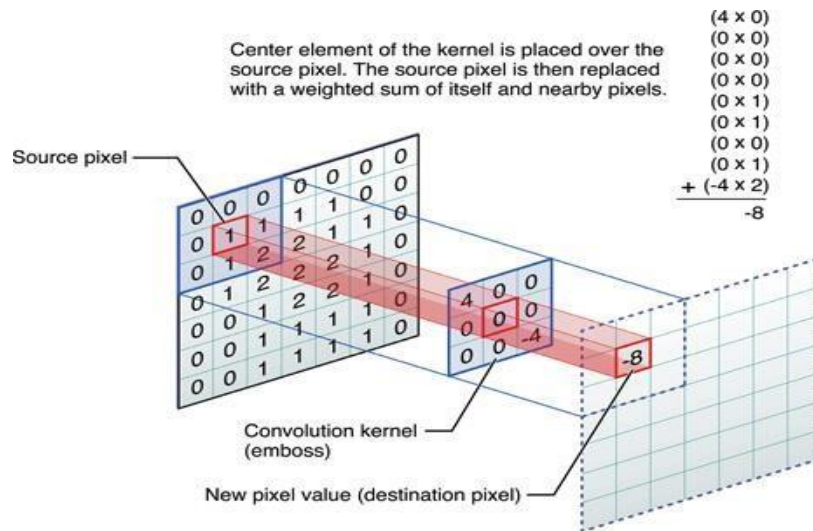


Fig3.10. Working of Convolution

Convolution works by replacing each pixel by a weighted sum of the pixels that surround it. The neural network has to learn the weights.

Now replace pixels with notes, and we have an idea for what we can do. If we make a stack of identical recurrent neural networks, one for each output note, and give each one a local neighborhood (for example, one octave above and below) around the note as its input, then we have a system that is invariant in both time and notes: the network can work with relative inputs in both directions.

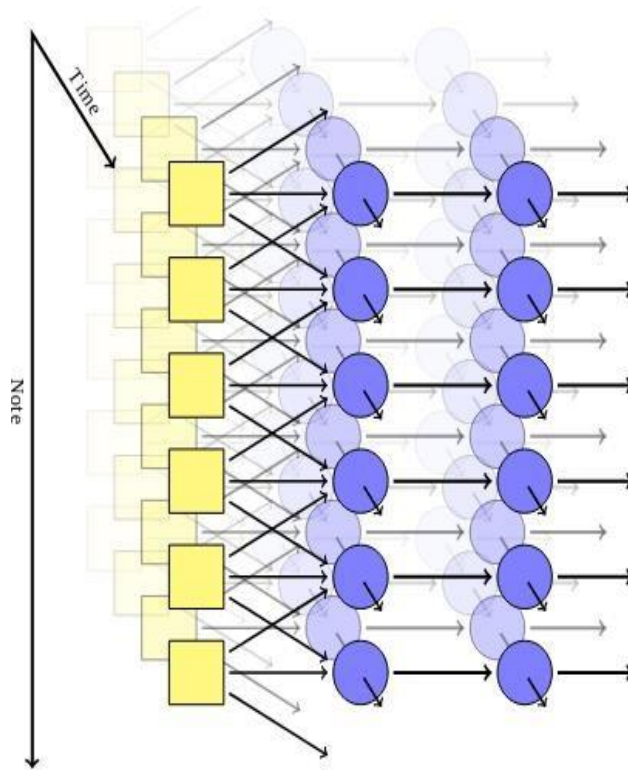


Fig.3.11. System invariant in both time and notes

Round the time axis here! Note that the time steps now exit the page, as does normal communication. You can think of the "flat" pieces as a copy of the basic RNN image from above. Also, we show each layer getting input in one note above and below. This simplification: the actual network receives input in 12 notes (the number of steps in the octave) on each side.

However, there is still a problem with this network. Normal communication allows patterns at a time, but we do they have no way of getting good chords: the output of each note is completely independent of all the output of the note. Here we can find inspiration in the RNN-RBM combination above: let the first part of our network face the moment, and let the second part create beautiful songs. But RBM offers one conditional distribution of multiple results, which is not compatible with using one network per note.

The solution I have decided to take with me is what I call “biaxial RNN”. The idea is that we have two axis (and one mock axis): there is a time axis and a note axis (and a pseudo-axis computation axis). Each recurring layer converts the input into the output, and sends a recurring connection with one of these axes. But there is no reason why everyone should post a connection with the same axis.

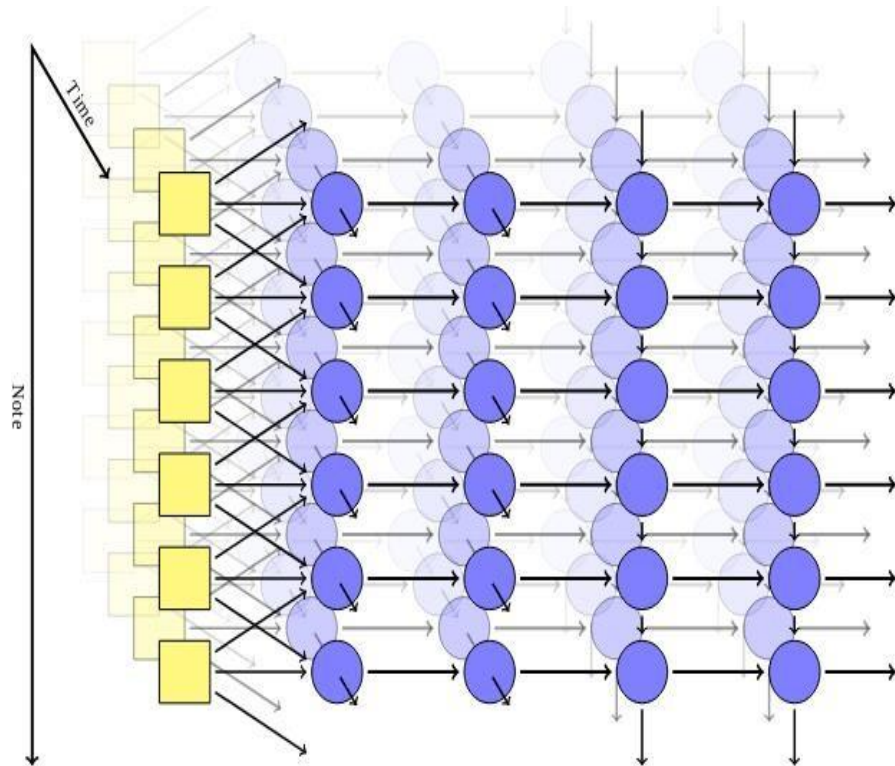


Fig3.12. Patterns in time and note space

Having patterns in time and note space without loss in invariance layers, on the other hand, have connections between notes, but are independent between time steps. Together, this allows us to have patterns both in time and in note-space without sacrificing invariance.

It's a bit easier to see if we collapse one of the dimensions as shown in the following figure:

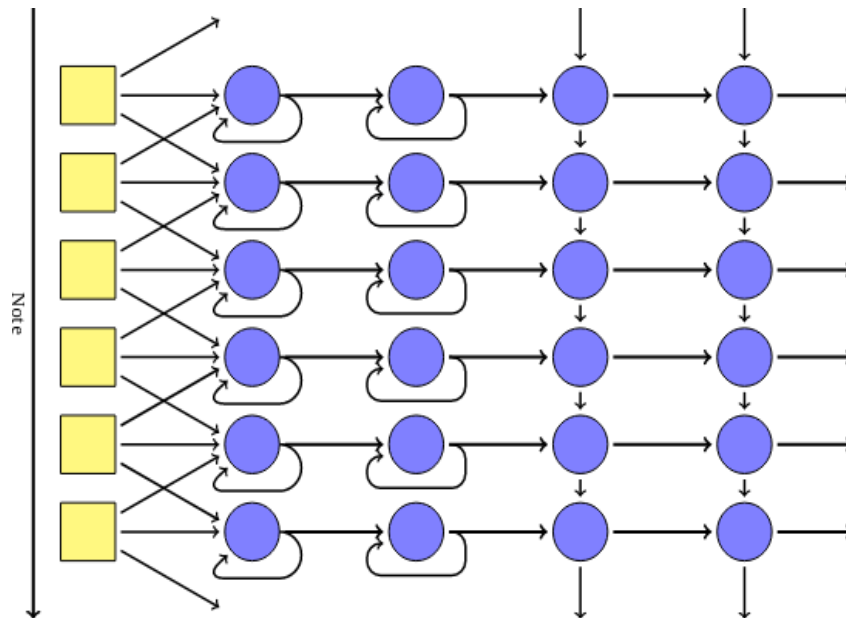


Fig3.13. Considering time connections as loops

Now the time connection is shown as loops. It is important to remember that loops are always delayed by one step: output by t is part of input by $t + 1$.

First, we can add the first axis layer at each step of time: (number in brackets the number of elements in the input vector corresponding to each component)

- Position [1]: MIDI note value for current note. It is used to get a vague idea of how high or low a given note is, to allow for differences (such as the idea that low notes are usually songs, high notes are usually musical).
- Pitchclass [12]: Will be 1 instead of current note, starting at A with 0 and increasing by 1 in each step, and then 0 in all others. Used to allow selection of standard songs (i.e. more often with a C-string than a larger E-flat chord)
- Previous Location [50]: Provides the context of the surrounding notes in the last step, one octave on each side. The value of point 2 (+12) is 1 if the note removes the current note being played in the last step, and 0 if it is not played. The value of 2 (+12) + 1 is 1

if that note is mentioned in the last step, and 0 if not specified. (So if you play a note and hold it, first you have 1 in both, second you get first. If you repeat the note, second you will have 1 both times.)

- Previous Content [12]: The reference value i will be the number of times any note x where $(x\text{-pitchclass}) \bmod 12$ was last played. So if the current note is C and there was a final step of 2 E, the value in point 4 (as E is 4 half the step above C) would be 2.

- Beat [4]: Basically a double representation of position within a scale, takes 4/4 time. Since each row is a slow insert, and each column is a step in time, it basically repeats the following pattern:

0101010101010101

0011001100110011

0000111100001111

0000000011111111

However, it is rated at $[-1, 1]$ instead of $[0,1]$.

Then there is the first hidden LSTM stack, which includes LSTMs with a recurring connection near the time axis. The end-time axis layer produces a specific note form that represents any time patterns. The second LSTM stack, which repeats near the axis of the note, then scans from the lower notes to the higher notes. In each step of the note (equal to the steps of time) is obtained as an intervention

- The vector status of the corresponding note of the previous LSM stack
- the value (0 or 1) of the previous note (lower step) selected to play (based on the previous step of the note, starting with 0)
- the value (0 or 1) selected from the previous note (lower step) for display (based on the previous note step, from 0)

After the last LSTM, there is a simple, repetitive layout layer that produces 2 values:

- Play Opportunities, which is the opportunity for this note to be selected for play

- Opportunity Optimization, which is the chance that a note is mentioned, when watching a game. (This is only used to determine duplicate of managed notes.)

The model is based on Theano, a Python library that makes it easy to generate fast neural networks by integrating a network into GPU-prepared code and automatically calculating gradients. We can install and use it according to the instructions.

to install sudo pip --upgrade theano

Sudo pip install numpy scipy theano-lstm python-midi

We can feed on a randomly selected collection of short music parts during training. We next calculate the cross-entropy of all the output options, which is an excellent technique to discover the best output options when considering the output potential. We attach it as a cost to the AdaDelta developer and allow it to increase our weight after some deceit using logarithms to make the jokes less likely to be stupid, followed by denial to be a reduction problem.

We may accelerate training by utilising the fact that we already know which product to select for each phase. Basically, we can start by combining all of the notes and training the time axis layers, then reorganise the output so that all of the times are combined and all of the note's axis layers are trained. This allows us to take advantage of the GPU's ability to duplicate huge matriculants.

We can use something called stop reading to keep our model from becoming too symmetrical (which can mean reading certain parts of specific pieces instead of complete patterns and features). Using stop reading means randomly moving a portion of the hidden notes in each layer throughout each training step. This prevents nodes from being drawn into each other's weak dependencies, and instead promotes specialisation. (We may do this by repeating the question with the results of each layer.) By setting the zero exit in a certain time step, notes are "erased.").

Unfortunately, we are unable to successfully combine everything during the training. For each step of time, we must first apply one tick to the time axis layers, then run all the repeated sequences of note axis layers to determine which input should give you the time axis layers the next marker. This slows down the process. In addition, we should

add a corrective element to our responses by pausing during training. In practise, this means that the output 0.5 for each node is 0.5. This prevents the network from overheating as a result of the large number of active nodes.

CHAPTER-4 PERFORMANCE ANALYSIS

We are using the open-sourced data available on the ABC version of the Nottingham Music Database. It contains more than 1000 folk tunes, the vast majority of which have been converted to ABC notation.

The data is currently in a character-based categorical format. In the data processing stage, we need to transform the data into an integer-based numerical format, to prepare it for working with neural networks.

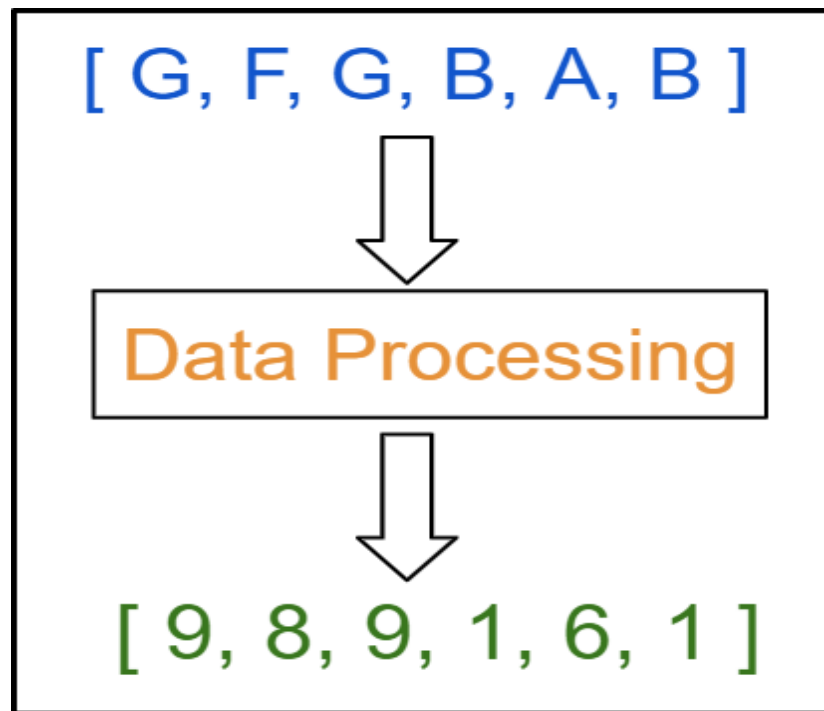


Fig4.1. Data Processing Overview

Here each character is mapped to a unique integer. This can be achieved using a single line of code. The 'text' variable is the input data.

To process the output at each timestamp, we create a time distributed dense layer. To achieve this we create a time distributed dense layer on top of the outputs generated at each timestamp.

The output from the batch is passed to the following batch as input by setting the parameter stateful to true. After combining all the features, our model will look like the overview depicted in figure below.

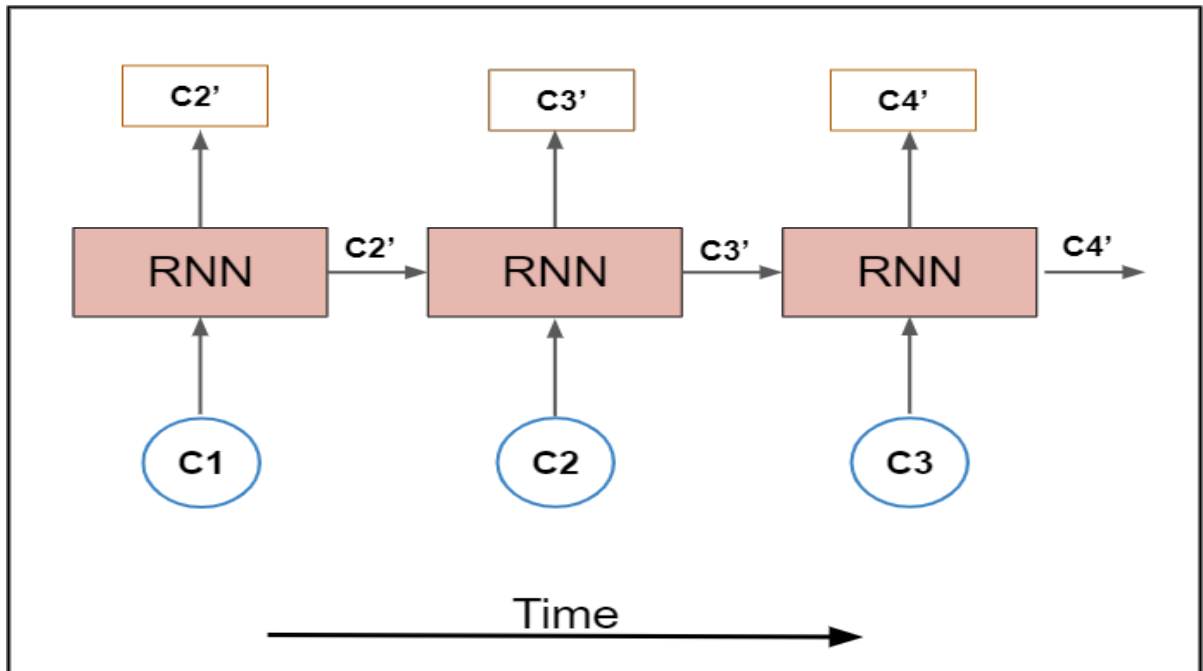


Fig4.2. Overview of model

Dropout layers are a practise that reduces a portion of the input units to zero in each update during training to prevent over-fitting. The fraction is decided by the base and the parameter used. Music production is a problem of dividing multiple categories, each of which has a distinct character from the input data. As a result, we apply a softmax layer over our model and a cross-entropy phase as a loss function.

Every class has opportunities thanks to this layer. We choose the ones with the greatest potential from the list of opportunities.

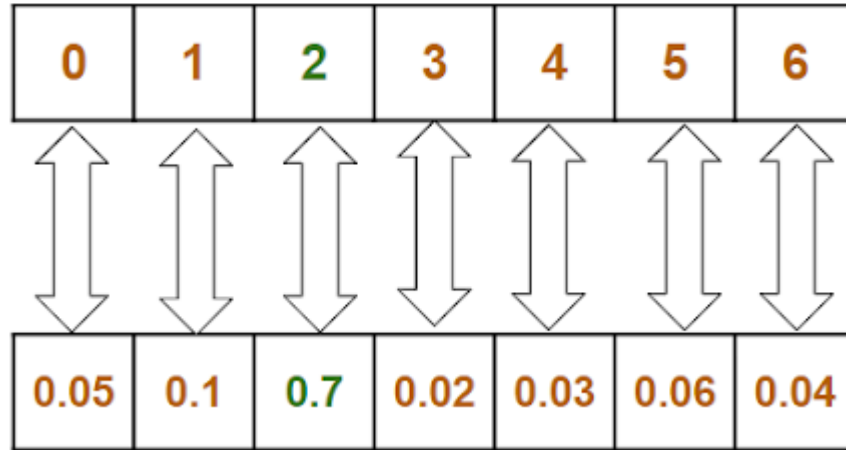


Fig4.3. Probability of each class

To optimize our model, we use Adaptive Moment Estimation, also called Adam as it is a very good choice for RNN.

Layer (type)	Output Shape	Param #
embedding_15 (Embedding)	(16, 64, 512)	44032
lstm_45 (LSTM)	(16, 64, 256)	787456
dropout_45 (Dropout)	(16, 64, 256)	0
lstm_46 (LSTM)	(16, 64, 256)	525312
dropout_46 (Dropout)	(16, 64, 256)	0
lstm_47 (LSTM)	(16, 64, 256)	525312
dropout_47 (Dropout)	(16, 64, 256)	0
time_distributed_15 (TimeDis	(16, 64, 86)	22102
activation_15 (Activation)	(16, 64, 86)	0
=====		
Total params: 1,904,214		
Trainable params: 1,904,214		
Non-trainable params: 0		

Fig4.4. Adaptive Moment Estimation used

So far, we've created an RNN model and trained it using our input data. During the training phase, this model learned the inclusion data patterns. This model will be referred to as a 'trained model.'

The size of the input in the trained model is the size of the collection. And one of the factors in making music using machine learning is the input size. As a result, we created a new model that is similar to the previous one, but with a one-letter input size (1,1). We load weights from a professional model into this new model to duplicate the features of a professional model.

We load the weights of the trained model to the new model.

Layer (type)	Output Shape	Param #
embedding_2 (Embedding)	(1, 1, 512)	44032
lstm_6 (LSTM)	(1, 1, 256)	787456
dropout_6 (Dropout)	(1, 1, 256)	0
lstm_7 (LSTM)	(1, 1, 256)	525312
dropout_7 (Dropout)	(1, 1, 256)	0
lstm_8 (LSTM)	(1, 1, 256)	525312
dropout_8 (Dropout)	(1, 1, 256)	0
time_distributed_2 (TimeDist	(1, 1, 86)	22102
activation_2 (Activation)	(1, 1, 86)	0
=====		
Total params: 1,904,214		
Trainable params: 1,904,214		
Non-trainable params: 0		

Fig4.5 Weights of trained model loaded to new model

In the process of music generation, the first character is chosen randomly from the unique set of characters, the next character is generated using the previously generated character and so on. With this structure, we generate music.



Fig4.6 Black-box like model of the project flow

RESULTS

```
import os
i = []
counter = 0
shapes = []
folder_path = "/content/data"
for midifile in os.listdir(folder_path):
    #print(midifile)
    print(counter)
    counter = counter + 1
    t,j=getnotes_chords_rest(folder_path + "/" + str(midifile))
    i.append(t)
    shapes.append(np.shape(t))

24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
/usr/local/lib/python3.7/dist-packages/numpy/core/_asarray.py:83: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of
return array(a, dtype, copy=False, order=order)
53
54
```

Fig4.7. No of midi files

```
87
88
89
90
91
```

```
notes = []
for k in i:
    for e in k:
        if e != []:
            for n in e:
                notes.append(n)
print(np.shape(notes))
```

(59735,)

Fig4.8. Counting no. of nodes

```
[7] seq_len=50
notes = notes[:5000]
```

```
[8] mean = 86.0
n_vocab = 173
seq_len = 50
int_to_note = {-0.49710982658959535: '0', -0.4913294797687861: '0.1', -0.48554913294797686: '0.2', -0.4797687861271676: '0.2.5', -0.47398843930635837: '0.3', -0.46820809248
note_to_int = {'0': -0.49710982658959535, '0.1': -0.4913294797687861, '0.2': -0.48554913294797686, '0.2.5': -0.4797687861271676, '0.3': -0.47398843930635837, '0.3.7': -0.46
```

```
pitch_names=sorted(set(ele for ele in notes))
n_vocab=len(pitch_names)
print(n_vocab)
```

191

Fig4.9. Counting no. of pitches

```
note_to_int = dict((note,(number-mean)/n_vocab) for number,note in enumerate(pitch_names))
##PREPAING THE INPUT AND THE OUTPUT DATA FOR THE MODEL
net_input = []
net_output = []
#notes = i[0]
for i in range(len(notes)-seq_len):
    temp_in = notes[i:i+seq_len]
    temp_out = notes[i+seq_len]

    net_input.append([note_to_int[ch] for ch in temp_in])
    net_output.append(note_to_int[temp_out])
#print(np.shape(net_input),np.shape(notes))
n_patterns = len(net_input)
# reshape the input into a format compatible with LSTM layers
net_input = np.reshape(net_input, (n_patterns, seq_len, 1))
# normalize input
#net_input = net_input
#net_output = np.array(net_output)
net_output = (np.array(net_output)*n_vocab)+mean
#min_out = net_output.min()
#net_output = (net_output - min_out)*n_vocab
#print(np.shape((net_output + mean)*n_vocab))
net_output = np_utils.to_categorical(net_output)
print(np.shape(net_output))

(4950, 191)
```

Fig4.10. Input and output of data of model

```

from tensorflow import keras
from keras import optimizers
optimizer = keras.optimizers.Adam(lr = 0.00001)
model_compile(model,optimizer)
model_train(model,net_input, net_output, 30, 64,30,6401)

Epoch 2/30
78/78 [=====] - 59s 755ms/step - loss: 5.1410 - accuracy: 0.0402
Epoch 3/30
78/78 [=====] - 58s 744ms/step - loss: 4.8217 - accuracy: 0.0325
Epoch 4/30
78/78 [=====] - 59s 755ms/step - loss: 4.7202 - accuracy: 0.0362
Epoch 5/30
78/78 [=====] - 59s 750ms/step - loss: 4.6728 - accuracy: 0.0325
Epoch 6/30
78/78 [=====] - 59s 749ms/step - loss: 4.6575 - accuracy: 0.0354
Epoch 7/30
78/78 [=====] - 58s 739ms/step - loss: 4.6212 - accuracy: 0.0388
Epoch 8/30
78/78 [=====] - 57s 736ms/step - loss: 4.6166 - accuracy: 0.0380
Epoch 9/30
78/78 [=====] - 59s 754ms/step - loss: 4.6160 - accuracy: 0.0358
Epoch 10/30
78/78 [=====] - 58s 740ms/step - loss: 4.5969 - accuracy: 0.0343
Epoch 11/30
78/78 [=====] - 59s 749ms/step - loss: 4.5799 - accuracy: 0.0352
Epoch 12/30
78/78 [=====] - 58s 745ms/step - loss: 4.5739 - accuracy: 0.0335
Epoch 13/30
78/78 [=====] - 58s 747ms/step - loss: 4.5592 - accuracy: 0.0329
Epoch 14/30
78/78 [=====] - 58s 742ms/step - loss: 4.5648 - accuracy: 0.0362
Epoch 15/30
78/78 [=====] - 59s 758ms/step - loss: 4.5548 - accuracy: 0.0368
Epoch 16/30
78/78 [=====] - 58s 744ms/step - loss: 4.5538 - accuracy: 0.0358
Epoch 17/30
78/78 [=====] - 58s 740ms/step - loss: 4.5382 - accuracy: 0.0362
Epoch 18/30
78/78 [=====] - 58s 739ms/step - loss: 4.5447 - accuracy: 0.0358
Epoch 19/30
78/78 [=====] - 58s 744ms/step - loss: 4.5141 - accuracy: 0.0366
Epoch 20/30
78/78 [=====] - 58s 741ms/step - loss: 4.5284 - accuracy: 0.0329
Epoch 21/30
78/78 [=====] - 58s 746ms/step - loss: 4.5178 - accuracy: 0.0370
Epoch 22/30
78/78 [=====] - 58s 750ms/step - loss: 4.5179 - accuracy: 0.0352
Epoch 23/30
78/78 [=====] - 58s 747ms/step - loss: 4.4978 - accuracy: 0.0406
Epoch 24/30
78/78 [=====] - 58s 743ms/step - loss: 4.4918 - accuracy: 0.0402
Epoch 25/30
78/78 [=====] - 58s 749ms/step - loss: 4.4904 - accuracy: 0.0390
Epoch 26/30
78/78 [=====] - 57s 735ms/step - loss: 4.4744 - accuracy: 0.0438
Epoch 27/30
78/78 [=====] - 58s 750ms/step - loss: 4.4491 - accuracy: 0.0479
Epoch 28/30
78/78 [=====] - 58s 740ms/step - loss: 4.4294 - accuracy: 0.0485
Epoch 29/30
78/78 [=====] - 59s 751ms/step - loss: 4.4021 - accuracy: 0.0515
Epoch 30/30
78/78 [=====] - 58s 743ms/step - loss: 4.3894 - accuracy: 0.0491

```

Fig4.11. Model Training

```

] print(music_out)

['A4', 'G4', 'A4', 'G4', 'D3', 'G4', 'B-4', 'G4', 'B-4', 'G4', 'D5', 'A4', 'G4', 'G4', 'C5', 'C5', 'F4', 'G5', 'G5', 'C2', 'G5', 'C5', 'G5', 'C5', 'C5', '

```

Fig4.12. Musical Notes as Output


```
model.summary()

Model: "sequential"
-----
Layer (type)                Output Shape              Param #
-----
bidirectional (Bidirectiona (None, 50, 512)          528384
l)

dropout (Dropout)           (None, 50, 512)          0

seq_self_attention (SeqSelf (None, 50, 512)          32833
Attention)

lstm_1 (LSTM)               (None, 50, 128)          328192

dropout_1 (Dropout)         (None, 50, 128)          0

flatten (Flatten)          (None, 6400)              0

dense (Dense)               (None, 256)               1638656

dropout_2 (Dropout)         (None, 256)               0

dense_1 (Dense)             (None, 191)               49087

activation (Activation)     (None, 191)               0

-----
Total params: 2,577,152
Trainable params: 2,577,152
Non-trainable params: 0
```

Fig4.13. Model Summary

CHAPTER-5 CONCLUSIONS

CONCLUSION

After designing the model and analysing some of the outputs, we have concluded that LSTM is a good candidate for automatic generation of music by just feeding some small input samples to it.

But, there is surely a lot of room for improvement in what we have at our hands right now. The music generated by the model was very interesting and listenable but the quality of the music can surely be improved upon.

FUTURE SCOPE

We have intentions to increase the breadth of inputs our model can take and output corresponding music, since we are limited to only classical music as of now.

We would also like to increase the sequence length to enhance the model. (Sequence length means the length of sequence input in model.)

We would also like to improve the quality of music, as it is listenable and interesting, but can surely be worked upon.

REFERENCES

- [1] <https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/>
- [2] <https://www.danieldjohnson.com/2015/08/03/composing-music-with-recurrent-neural-networks/>
- [3] <https://in.mathworks.com/help/deeplearning/ug/long-short-term-memory-networks.html>
- [4] <https://towardsdatascience.com/generating-music-using-deep-learning-cb5843a9d55e>
- [5] <https://towardsdatascience.com/music-generation-through-deep-neural-networks-21d7bd81496e>
- [6] <https://machinelearningmastery.com/gentle-introduction-long-short-term-memory-networks-experts/>
- [7] <https://www.sciencedirect.com/>
- [8] Alexander Agung Santoso Gunawan, Ananda Phan Iman, Derwin Suhartono, “Automatic Music Generator Using Recurrent Neural Network”, International Journal of Computational Intelligence Systems Vol. 13(1), 2020, Published by Atlantis Press SARL, pp. 645-654
- [9] Alexandru-Ion Marinescu, “generating classical music using recurrent neural networks”
- [10] 23rd International Conference on Knowledge-Based and Intelligent Information & Engineering Systems, 2019, Published by Elsevier B.V.
- [11] Qibin Lou, “Music Generation using Neural Networks”, Stanford University

APPENDICES

Code1. Note to Integer Conversion

```
##DICTIONARY APPROACH
note_to_int = dict((note, (number-mean)/n_vocab) for number,note in
enumerate(pitch_names))
##PREPAING THE INPUT AND THE OUTPUT DATA FOR THE MODEL
net_input = []
net_output = []
#notes = i[0]
for i in range(len(notes)-seq_len):
    temp_in = notes[i:i+seq_len]
    temp_out = notes[i+seq_len]

    net_input.append([note_to_int[ch] for ch in temp_in])
    net_output.append(note_to_int[temp_out])
#print(np.shape(net_input),np.shape(notes))
n_patterns = len(net_input)
# reshape the input into a format compatible with LSTM layers
net_input = np.reshape(net_input, (n_patterns, seq_len, 1))
# normalize input
#net_input = net_input
#net_output = np.array(net_output)
net_output = (np.array(net_output)*n_vocab)+mean
#min_out = net_output.min()
#net_output = (net_output - min_out)*n_vocab
#print(np.shape((net_output + mean)*n_vocab))
net_output = np_utils.to_categorical(net_output)
print(np.shape(net_output))
```

Code2. Training Model

```
def get_model():
    model=Sequential()
    #model.add(Embedding)

model.add(Bidirectional(LSTM(256,return_sequences=True),input_shape=(net_in
put.shape[1], net_input.shape[2])))
    model.add(Dropout(0.3))
    model.add(SeqSelfAttention(attention_activation = "sigmoid"))
    model.add(LSTM(128, return_sequences=True))
    model.add(Dropout(0.3))
    model.add(Flatten())
    model.add(Dense(256, activation='relu'))
    model.add(Dropout(0.3))
    model.add(Dense(n_vocab))
    model.add(Activation('softmax'))
    return model

def model_compile(model,optimizer):
    model.compile(loss='categorical_crossentropy', optimizer=optimizer,
metrics = ['accuracy'])

def model_train(model,net_input,
net_output,iterations,batch_size,period,version):
    filepath = "/home/beyond100000/Documents/RNN_MUSIC/weights-
{epoch:02d}"+str(version)+".hdf5"
    checkpoint = ModelCheckpoint(filepath,monitor='loss',
verbose=0,
save_best_only=True,
mode='min',
period=period)
    model.fit(net_input, net_output, epochs=iterations,
batch_size=batch_size, callbacks=[checkpoint])
```

Code3. Music note generation

```
## MUSIC NOTE GENERATION
length = 50 ###instead of len(net_input)
start = np.random.randint(0, length-1)
print(start)
#int_to_note = dict(((number-mean)/n_vocab, note) for number, note in
enumerate(pitch_names))
#note_to_int = dict((note, (number-mean)/n_vocab) for number,note in
enumerate(pitch_names))
music_len = 500
music_out = []
#pattern = net_input[start]
#test_start = pattern
pattern = np.random.rand(50,1)
#print(pattern)
for i in range(music_len):
    if i%100 == 0 :
        print(i)
        #print(result)
    pred_init = np.reshape(pattern, (1, seq_len, 1))
    #pred_init = (pred_init - mean)/ n_vocab

    pred = model.predict(pred_init, verbose = 0)
    #print(pred)
    index = np.argmax(pred)
    #print(np.argmax(pred))
    index = (index - mean)/n_vocab
    result = int_to_note[index]
    music_out.append(result)
    #print(result)
    pattern = list(pattern)
    pattern.append(index)
    pattern = pattern[1:len(pattern)]
```

Code4. Generation node and chord object based on value of generated model

```
offset = 0
music = []
# create note and chord objects based on the values generated by the model
for pattern in music_out:
    #print(pattern)
    # pattern is a chord
    if ('.' in pattern) or pattern.isdigit():
        notes_in_chord = pattern.split('.')
        notes = []
        for current_note in notes_in_chord:
            new_note = note.Note(int(current_note))
            new_note.storedInstrument = instrument.Piano()
            notes.append(new_note)
        new_chord = chord.Chord(notes)
        new_chord.offset = offset
        music.append(new_chord)
    # pattern is a note
    else:
        new_note = note.Note(pattern)
        new_note.offset = offset
        new_note.storedInstrument = instrument.Piano()
        music.append(new_note)
# increase offset each iteration so that notes do not stack
offset += 0.5
```