MUSIC GENERATION USING NEURAL NETWORKS(LSTM).

BY:

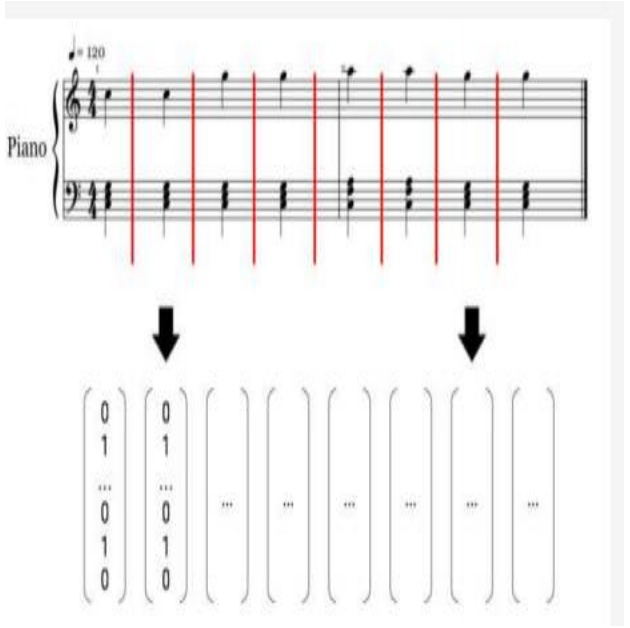
SAHIL KUMAR:2K18/CO/307

SAURABH SINGH: 2K18/CO/329

OBJECTIVE:

- The Objective of our project is to generate melodious music with the techniques of neural networks using LSTM , which should be similar to music created by music composer.

- This project aims to apply neural network algorithms to generate music given a repository of music to train on.



The development of this project had the following structure:

- Preparing the input data – decision the features take into account.
- Building the model and training on the input data
- Generating music with the model and evaluation of results

MOTIVATION:

There have been a lot of articles, information on how to generate text using neural networks but a quite few information on how to create music.

- The motivation is in using the now widely available corpus to learn musical styles automatically and to generate new musical content based on this.

- To be helpful to the people , who does not know too much about fundamentals of music to be able to generate music by themselves.

- From a more practical point of view, many applications can be imagined. For example, we could combine it with computer vision to generate background music corresponding to the mood of a film. More artistically, we could imagine a use case for a form of human-machine collaborative composition.

TOOLS USED:

- Keras.
- Sequential model.
- Music21 toolkit: for processing MIDI files.

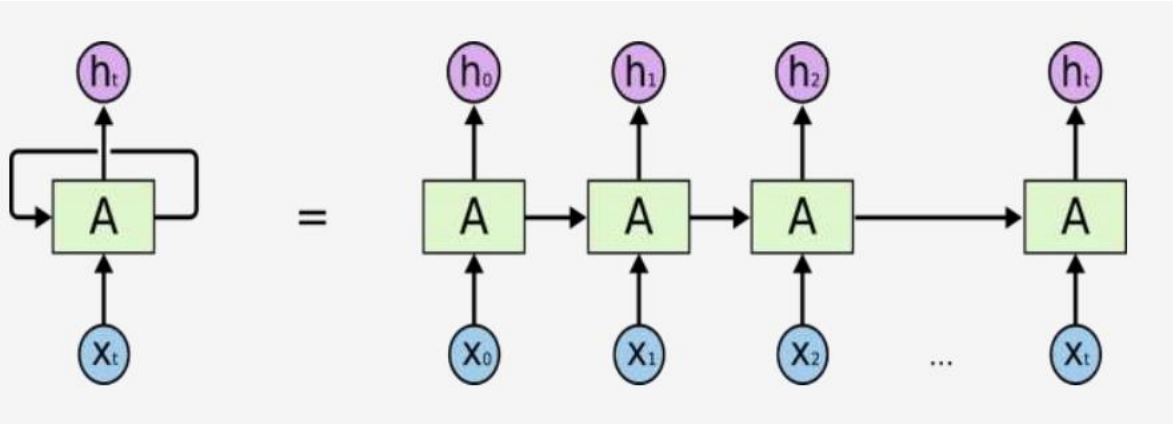
WHAT ARE ARTIFICIAL NEURAL NETWORKS?

- It is biologically-inspired programming paradigm which enables a computer to learn from observational data.

- It resembles human brains mainly these two ways:

  - A neural network acquires knowledge through learning.
  - A neural network's knowledge is stored within the interconnection strengths known as synaptic weight.

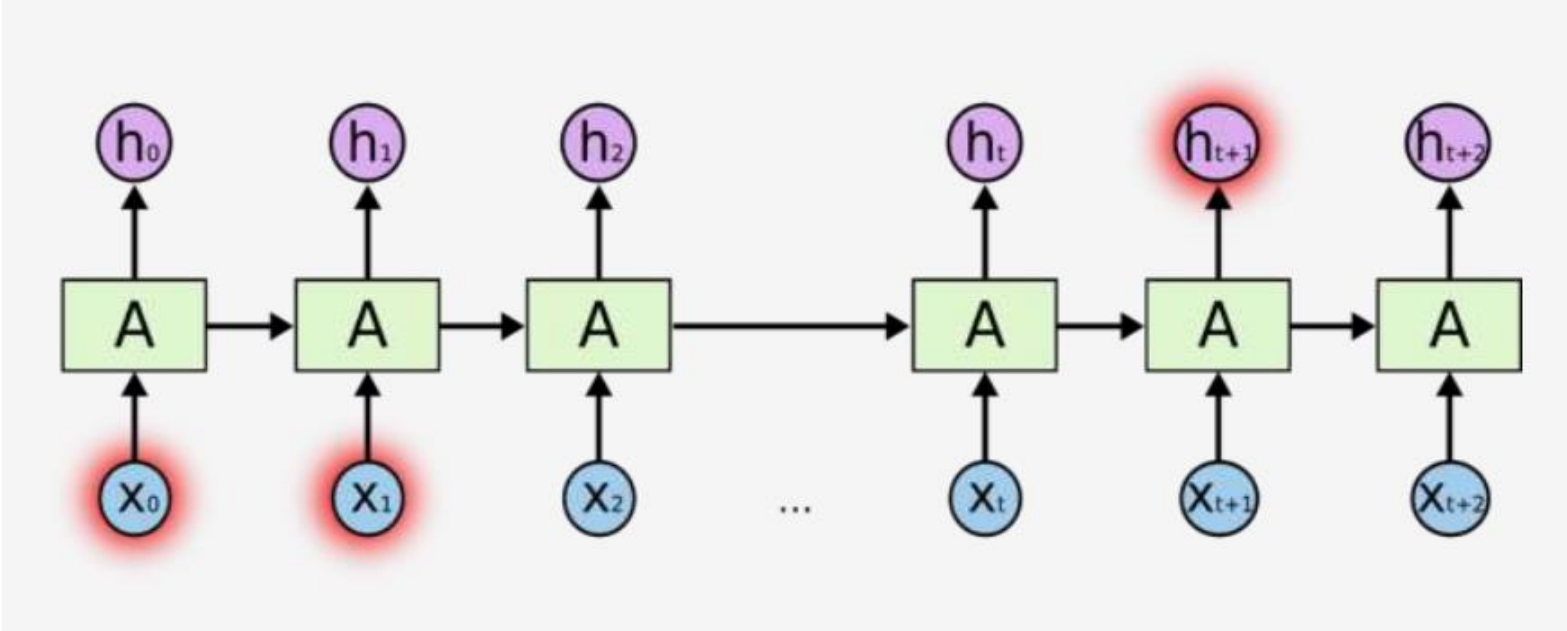
CAN WE USE RNN?



- RNNs have loops.
- Allows persistence of information.
- Each neuron accepts the input from previous layer as well as from previous neuron in the same layer.

Drawback: They cannot retain information for long periods of time.

CAN WE MAKE IT REMEMBER FOR LONG PERIOD?



Yes we can! By the use of

LSTM networks.

WHY LSTM,NOT RNN

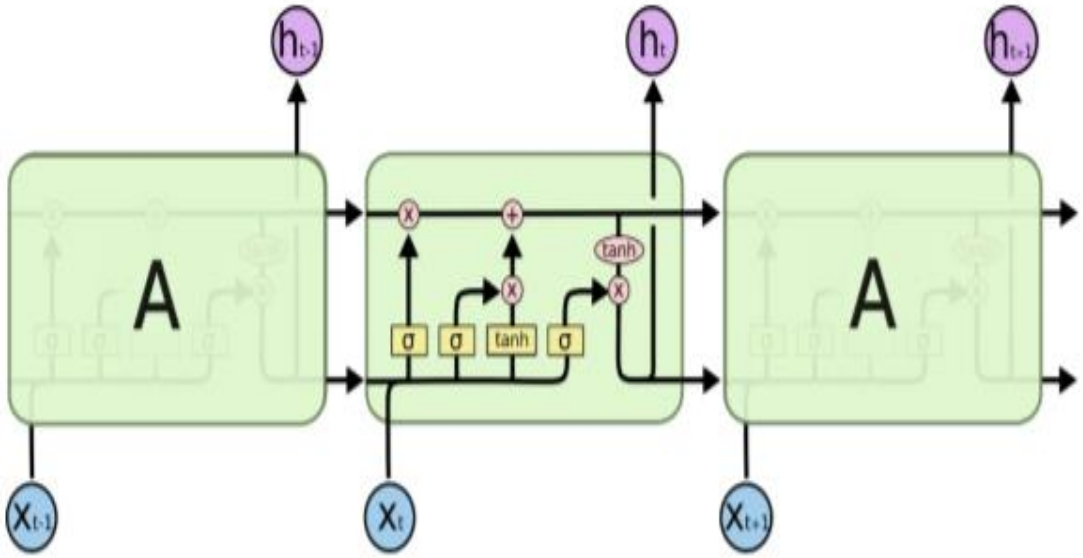
- Recurrent neural networks experience a serious problem known as vanishing or exploding gradient problem[1]. The problem is that they might not be able to connect information from several previous steps to the present step. This phenomenon was explored by Hochreiter and Bengio[2]. They proposed a solution called LSTM (Long short-term memory) cells which are capable of remembering long-term dependencies which are used in this paper.
- However, when information spans long intervals, RNN start having issues learning from them where it is shown that gradient descent becomes increasingly inefficient as the time span of dependencies increase. LSTMs aim to overcome exactly this problem .The main difference comes from the structure of the cell of the LSTM being more complex as we can see in the following figures.

MODEL USED:

- The model that we have used in our project is :

Long Short Term Memory networks-usually just called “LSTMs”-are a special kind of recurrent neural network(RNN).

They are capable of learning long term dependencies.

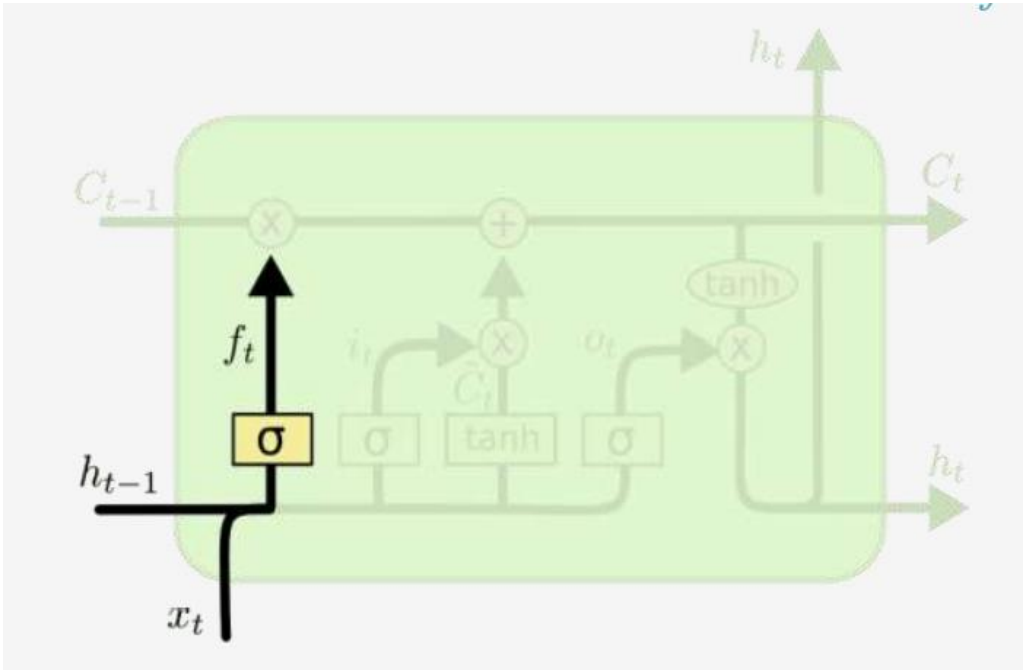


LSTM Model

WHAT ARE LSTM MODEL?

- Long Short Term Memory
- Special type of RNN.
- Capable of learning long term dependencies.
- It is well-suited to learn from experience to classify, process and predict time series when there are very long time lags of unknown size between important events.
- Explicitly designed to avoid the long-term dependency problem

STEP 1: FORGET GATE LAYER



- The first step in LSTM is to decide what information we're going to throw away from the cell state.
- This decision is made by a sigmoid layer called the "forget gate layer".

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

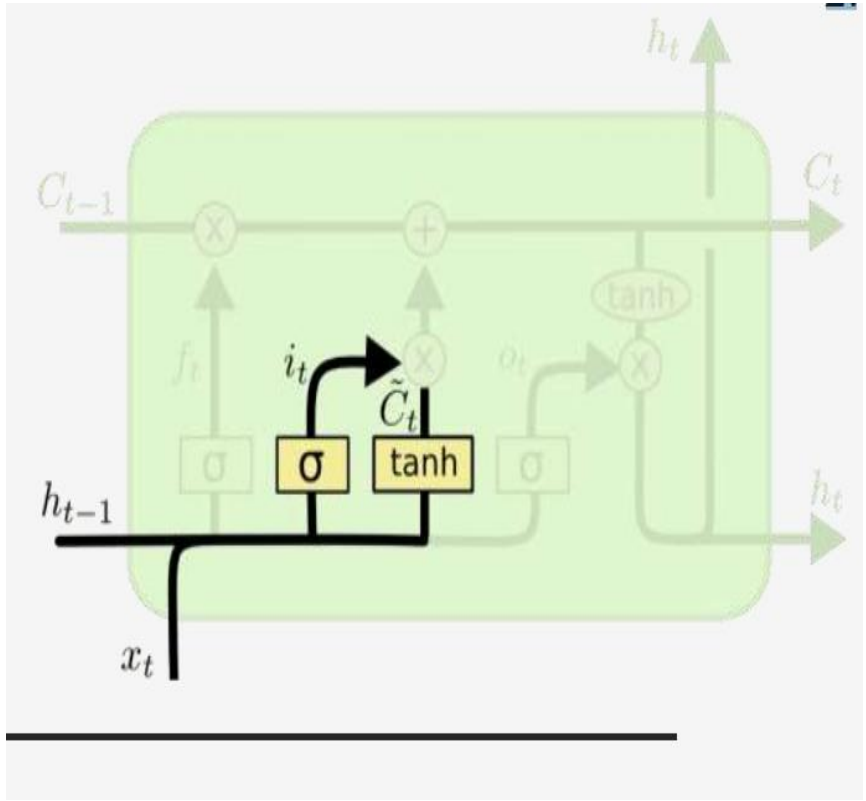
Where :

f_t, h_{t-1}, x_t – as defined to the right

W_f – the weights in input layer

b_f – the biases in input layer

STEP 2: NEW INFORMATION



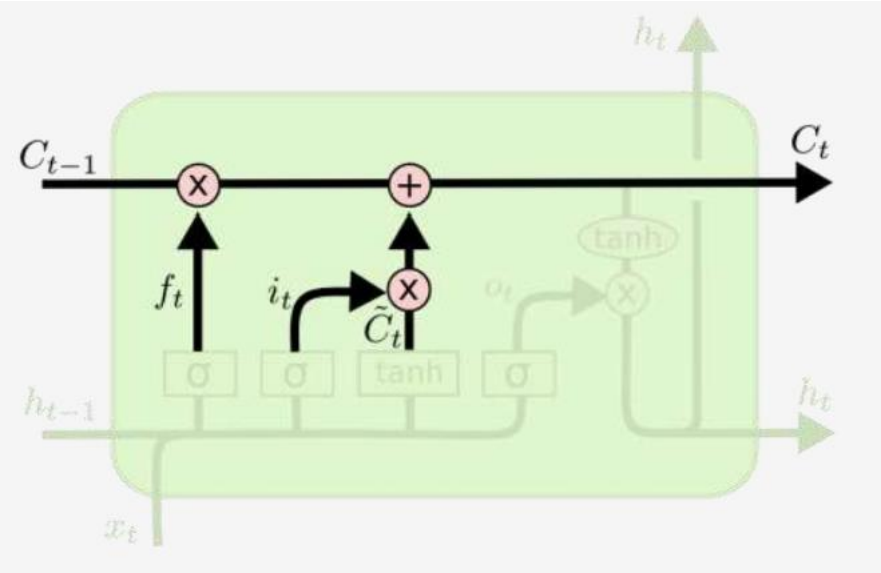
- Decide what new information we're going to store in the cell state.
- This has two parts:

1. A sigmoid layer called the "input gate layer" decides which values we'll update.
2. A tanh layer creates a vector of new candidate values, C_t that could be added to the state

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Where :
$i_t, h_{t-1}, x_t, \tilde{C}_t$ – as defined to the right
W_i – the weights in layer i
b_i – the biases in layer i

STEP 3: COMBINATION OF PREVIOUS STEPS



We update the new cell state by multiplying old cell state by f_t (forgetting the information designated in step 1) and then adding $i_t * C_t$ which are the new candidate values scaled by how much was decided to update each state value. (from step 2)

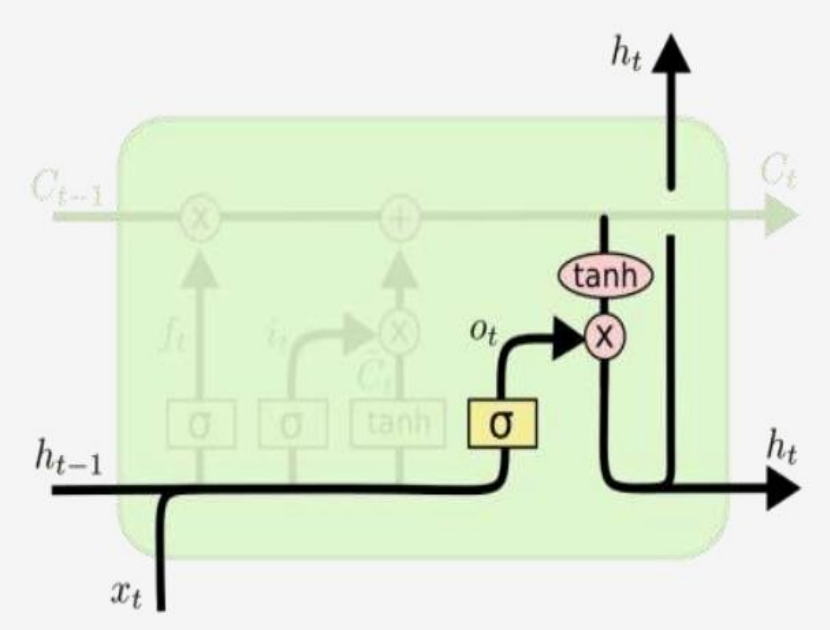
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Where :

$i_t, f_t, \tilde{C}_t$ – as previously defined

C_{t-1} – as defined to the right

STEP 4: OUTPUT



- we need to decide what we're going to output.
- Sigmoid layer decides what parts of the cell states we are going to output.
- The updated cell state vector is passed through a tanh layer and multiply it by the output of the sigmoid gate.

$$o_t = \sigma(W_o \,.\, [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * \tanh(C_t)$$

Where :

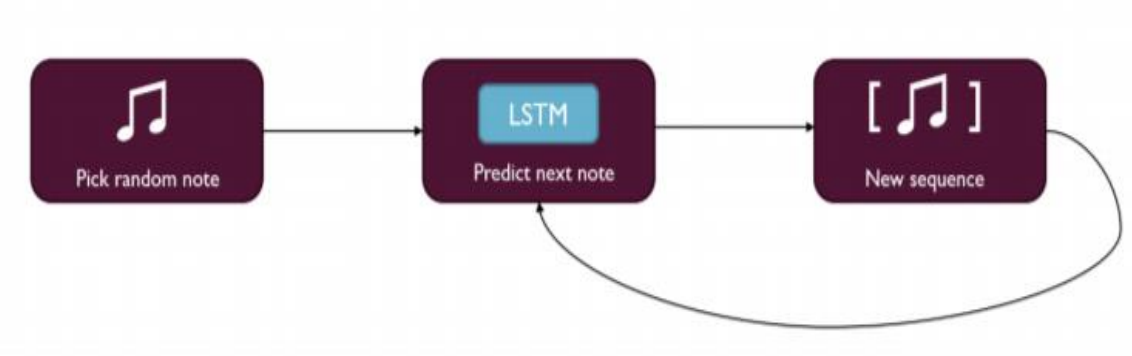
o_t, h_{t-1}, x_t – as defined to the right

W_o – the weights in output layer

b_o – the biases in output layer

C_t – as previously defined

- We have used an LSTM to predict what note comes after an input sequence of notes.



Music Generation system

A random note is fed into the model to predict the next note, we then use the output to feed it back into the trained model to generate the following one, etc. until we have a sequence of the desired length.

OUR LSTM MODEL DESIGN:

- The LSTM network is trained to acquire the knowledge of probability of occurrence of a musical note at current time. The output of the network at a time step t, conditioned on the previous notes' state till time step t-100 are fed into the input unit to recall past details and structure of notes.

The LSTM layer depends on a selected input. Not all notes undergo the training process. Only some selected and specified notes are used to train this LSTM model, which are useful for effectively tuning the model, that result in efficient information gain. With these inputs, LSTM layer learns the mapping and correlation between notes and their projection.

- Next to the LSTM layer, Dropout layer is used to create generalizations in the model. Once the model has learned the probability distributions of notes and sequences, we must combine all the LSTM cells with each other. This gap is subordinated with the help of Dense layer. Dense layer ensures that the model is fully connected.

At the endpoint, the Activation layer is added to the model, which helps in deciding, which neurons (LSTM cells) should be activated and whether the information gained by the neuron is relevant, making activation function highly important in a deep neural network. After training the LSTM network, the model is ready to generate a new sequence of musical notes.

ROADMAP FOR THE PROJECT:

Preparing the data:

1. Music21 toolkit for python to convert midi files into a list of notes & chords

    a. Put all the notes in a sequential list by appending the pitch, offset, duration in string notation

    b. Put all the chords in sequential list by encoding id of every note in the chord together to form a single string

2. Mapping function from above categorical data to numerical data

3. Creating input & output to train the model:

    a. Choose a sequence length as input and the output will be the first note/chord that comes after that sequence

    b. We can try here different sequence lengths to see how it impacts the model

METHODOLOGY:

PREPARING THE DATA:

The data that we choose for training are midi files from Albeniz dataset .The reason for choosing this dataset is that it has a large repertoire of keyboard works ,holds less complexity in terms of nuances and “expressiveness” in the direct notation of the music mainly.

It is also worth noting that the filetype chosen was MIDI. This presents itself as the obvious choice since MIDI files are essentially a list of events organized by tracks, events which will later be read by a musicplayer adding a soundfont to the even to color the notes. The structure is as follows:

The structure of midi file is as follows:

	Chunk				
MIDI File	Type	Length	Data		
	MThd	6	<format>	<tracks>	<division>
	MTrk	<length>	<delta_time> <event> ...		

Indeed, MIDI files are structured in chunks and each chunk contains a fixed size type, fixed size length and fixed size (based on the length specified) data. Further, there are two types of chunks: header chunks (“MThd”) which will contain information on the entire file and track chunks (“MTrk”) which will contain information on a specific track.

STEPS IN PREPARING THE DATA:

1) The features that we are gonna to use as the input and output of our LSTM network are notes and chords.

2) Will convert midi files into a list of notes and chords.

3)Will put all the notes and chords into a sequential list and can create the sequences that will serve as input of our network.

4)Will create a mapping function to map from string based categorical data to integer based numerical data. This is done because neural network perform much better with integer based numerical data than string based categorical data.

5) We have use sequence of length of 100 notes/chords.It means to predict the next note ,network must has previous 100 notes to help make the prediction.

Training:

1. LSTM
    a. First layer should have same number of nodes as the number of outputs the system has so that it maps to one of our classes.

2. Dropout - to regularize by setting a fraction of input to 0 at each update during training to prevent overfitting
    a. Determine what fraction to drop.

3. Dense - fully connected neural net where each input node is connected to an output node

4. Activation - what activation function to use to calculate output.

5. Determine how many of each layer we need

6. Calculate loss for each iteration of training. Loss functions in Keras:

    a. Mean squared error - too many categories ?
    b. Mean absolute percentage error - too many categories ?
    c. Categorical Cross entropy

7. Get the weights to use for model.

STRUCTURE OF OUR MODEL:

Let's define the structure of our model:

- 3 LSTM layers
- 3 Dropout layers
- 2 Dense layers
- 1 Activation layer
  - Activation function: softmax ,

our activation function chosen is softmax rather than tanh, for the simple reason that softmax will provide us with the probability distribution of each target class over all possible target classes and this is required for the loss function we are going to choose.

We choose to use categorical cross-entropy as our loss function as it works with softmax and is one of the most used loss functions for multiclass classification tasks.

We choose 256 hidden units inside of our LSTM layers because the higher complexity will allow the model to learn efficiently as our number of classes is already quite high.

We keep our dropout rate at 0.3, which is pretty low, but there is not a lot of noise inside of our data since they are direct music tracks and as fairly consistent and in our case it would be better for the model to overfit rather than not learn properly.

We had trained our model for 50 epochs with each batch that is propagated through network containing 50 samples. The whole training took around 5 hours.

Testing:

1. Pick a random note from the list of notes as starting point, run generation code to get next sequence, then remove the first note from original sequence append the generated sequence and generate a following note, etc.

2. We generate 200 notes using the network since that is roughly 1 min of music and gives the network plenty of space to create a melody .For each note that we want to generate we have to submit a sequence to the network.

2. Map back from numerical data to categorical data(from integers to notes).

3. Convert back to midi file .

4. Play midi file (pygame has a module).

CONCLUSION:

Through this project , we are able to achieve the goal of designing a model which can be used to generate music and melodies automatically without any human intervention. The model is capable to recall the previous details and generate a sequence of music using a single layered LSTM model, proficient enough to learn harmonic and melodic note sequence from MIDI files. The model design is described with a perception of functionality and adaptability. Moreover, analysis of the model is also impersonated for better insights and understanding.

FUTURE WORK:

- Future work will aim to test how well this model scales on much larger dataset.

- We would like to observe effects on this model by adding more LSTM units and try different combinations of hyper parameters to see how well this model performs. We believe follow up research can optimize this model further with lots of computation.

LIST OF REFERENCES:

- [1] P. S. Yoshua Bengio and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," 1994.
- [2] S. H. Schmidhuber and Jürgen, "Long short-term memory," 1997.
- [3]A. Ç. Yiiksel , M. M. Karci and A. Ş. Uyar, "Automatic music generation using evolutionary algorithms and neural networks," 2011 International Symposium on Innovations in Intelligent Systems and Applications, Istanbul, 2011, pp. 354-358, doi: 10.1109
- [4]Alpern A. : Tecnique for Algorithmic Composition of Music. Hampshire College. (1995)
- [5]Worth P, .et al :Growing Music: Musical Interpretations of L-Systems Lecture Notes in Computer Science. (2005)
- [6]Kang, Semin & Ok, Soo-Yol & Kang, Young-Min. (2012). Automatic Music Generation and Machine Learning Based Evaluation. 10.1007/978-3-642-35286-7_55.
- https://www.researchgate.net/publication/302206040_Automatic_Music_Generation_and_Machine_Learning_Based_Evaluation
- https://ieeexplore.ieee.org/abstract/document/5946091
- https://arxiv.org/pdf/2002.00212.pdf