Music Generation

Amogh Rajesh Desai
PES1201700180

Siddhant Samyak
PES1201700247

Amit Kumar
PES1201701295

***Abstract*--This paper depicts the work and analysis of different ways of using deep learning (deep artificial neural networks) to generate musical content.**

I. INTRODUCTION

Music is the ultimate language. Many amazing composers throughout history have composed pieces that were both creative and deliberate. Composers such as Bach were well known for being very precise in crafting pieces with a great deal of underlying musical structure. Is it possible then for a computer to learn to create such a musical structure?

Inspired by research papers based on the generation of polyphonic music that seemed to have a melody and some harmonization[4], we decide to tackle the same problem. More recent work on polyphonic music modeling, centered around time-series probability density estimation, has met some partial success. In particular, there has been a lot of work based on Recurrent Neural Networks and other similar recurrent energy-based models. Our approach, however, is to come up with a neural network model that will be adequate for generating musical notes. We try to answer two main questions

1. Is there a meaningful way to represent notes in music as a vector? That is, does a method of characterizing the meaning of words like word2vec[6] translate to music?

2. Can we build interesting generative neural network architectures that effectively express the notions of harmony and melody? Most pieces have a main melody throughout the piece that it might expand on; can our neural network do the same?

II. BACKGROUND AND RELATED WORK

One of the earliest papers on deep learning-generated music, written by Chen et al [2], generates one music with only one melody and no harmony. The authors also omitted dotted notes, rests, and all chords. One of the main problems they cited is the lack of global structure in the music. This suggests that there are two main directions to improve upon 1. create music with musical rhythm, more complex structure, and utilizing all types of notes including dotted notes, longer chords, and rests. 2. create a model capable of learning long-term structure and possessing the ability to build off a melody and return to it throughout the piece Liu et al. [5] tackle the same problem but are unable to overcome either challenge. They state that their music representation does not properly distinguish between the melody and other parts of the piece and in addition do not address the full complexity of most classical pieces. They cite two papers that try to tackle each of the aforementioned problems. Eck et al. [3] use two different LSTM networks – one to learn chord structure and local note structure and one to learn long term dependencies in order to try to learn a melody and retain it throughout the piece. This allows the authors to generate music that never diverges far from the original chord progression melody. However, this architecture trains on a set number of chords and is not able to create a more diverse combination of notes.

III. TECHNIQUES INVOLVED

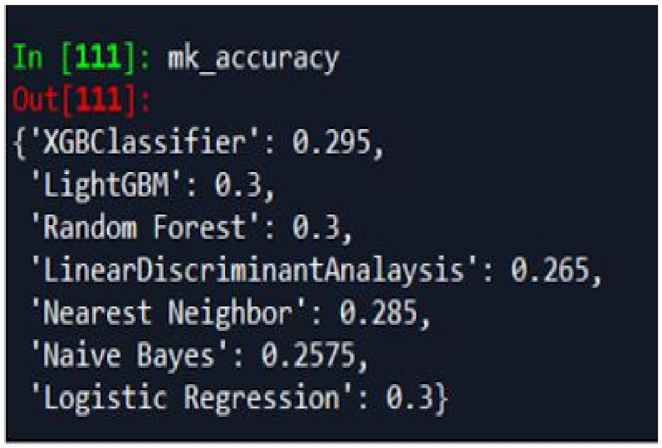
In our project, we will mainly tackle the problem of learning complex structure and rhythms based on

- Boulanger-Lewandowski et al. [1] which tries to deal with the challenge of learning a complex polyphonic structure in music. This model uses a recurrent temporal restricted Boltzmann machine (RBM) in order to model unconstrained polyphonic music. Using the RBM architecture allows them to represent a complicated distribution over each time step rather than a single token as in most character language models. This allows them to tackle the problem of polyphony in generated music.

- For a comparative analysis, we will have another model which will be a Sequential Stacked LSTM model sandwiching multiple Recurrent Neural Network layers and Dropout layers to prevent possible overfitting. LSTM is a type of recurrent neural network (proposed by Hochreiter and Schmidhuber, 1997) that can remember a piece of information and keep it saved for many timesteps.

IV. EXPLANATION FOR DEVISING THE ABOVE MENTIONED TECHNIQUES

Based on previous works, a group of 7 different models (classifiers) were trained and tested on the MIDI dataset for the purpose of musical notes generation. This was done so that the music generated by the models would be made in the same style as the composer the model was trained on. This is in the hope that the classifier would accurately predict the music generated by these models as being composed by the composer they were trained on. The following results show quantitatively the inferiority of Machine Learning (ML) classifiers:



```
In [111]: mk_accuracy
Out[111]:
{'XGBClassifier': 0.295,
 'LightGBM': 0.3,
 'Random Forest': 0.3,
 'LinearDiscriminantAnalaysis': 0.265,
 'Nearest Neighbor': 0.285,
 'Naive Bayes': 0.2575,
 'Logistic Regression': 0.3}
```

Fig 1: Classification Accuracies

Other stochastic models like the Markov model did not produce promising results.

We realized that musical notes generation is similar to text generation.

"As letters are used to generate words in a sentence, similarly the music vocabulary is used to generate music which is defined by the unique pitches in the notes list."

A. *Generative Model Selection*

There is a temporal long-term dependency on the data being fed to the model. Thus, deep learning models like Restricted Boltzmann Machine (RBM) and Long Short Term Memory (LSTMs) are favorable for this problem.

V. DATA

One of the primary challenges in training models for music generation is choosing the right data representation. We chose to focus on two primary types: midi files with minimal preprocessing and a "piano-roll" representation of midi files.

A. *What is MIDI?*

MIDI (Musical Instrument Digital Interface) is a protocol that allows electronic instruments and other digital musical tools to communicate with each other. Since a MIDI file only represents the player information, i.e., a series of messages like 'note on', 'note off'.

B. *Why MIDI?*

Compared to other formats, it is more compact, easy to modify, and can be adapted to any instrument.

We will use the music21 toolkit (a toolkit for computer-aided and musicology, MIT) to extract data from these MIDI files and in turn extract notes and chords present. Each input sequence will consist of 100 notes. The vector sequence is reshaped and normalized before feeding to the model.

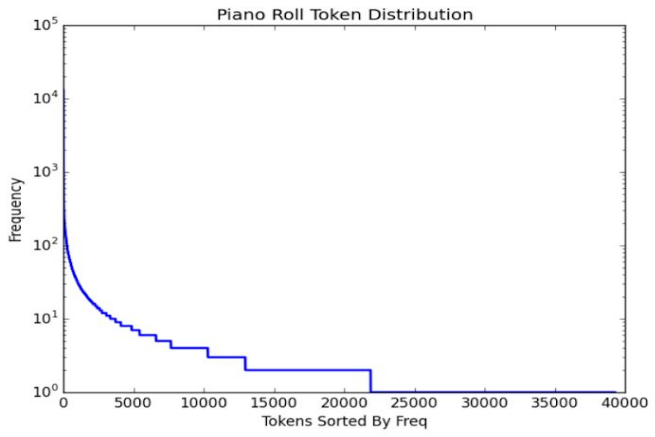


Fig 2: Frequency distribution of all the tokens in the dataset.

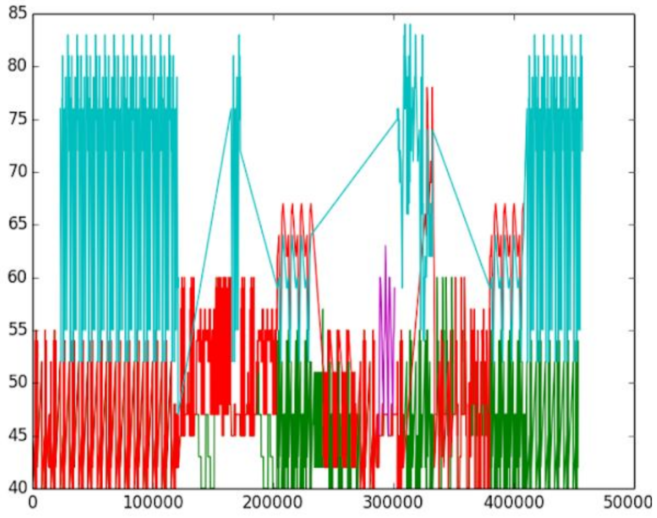


Fig 3: Visualisation of a MIDI file using music21 library. The turquoise color is the sweep picking in the song. Frequency vs Time.

VI. GENERATIVE MODELS

Generative Models specify a probability distribution over a dataset of input vectors. For an unsupervised task, we form a model for P(x), where x is an input vector. For a supervised task, we form a model for P(x|y), where y is the label for x. Like discriminative models, most generative models can be used for classification tasks. To perform classification with a generative model, we leverage the fact that if we know P(x|y) and P(y), we can use Bayes rule to estimate P(y|x).

$$P(\mathrm{A}|\mathrm{B}) = \frac{P(\mathrm{B}|A) \cdot P(A)}{P(B)}$$

Fig 4: Bayes Rule

VII. RESTRICTED BOLTZMANN MACHINES (RBM)

A simple Boltzmann machine is a bi-directionally connected network of stochastic processing units, that can be interpreted as a neural network model. A Restricted Boltzmann Machine, as suggested by Geoff Hinton is a *generative stochastic artificial neural network* that is capable of learning a probability distribution over its set of inputs. Simply put, the model automatically finds patterns in our data by reconstructing the input.

A. RBM Architecture

The RBM is a neural network with 2 layers, the visible layer, and the hidden layer. Each visible node is connected to each hidden node (and vice versa), but there are no visible-visible or hidden-hidden connections (the RBM is a complete bipartite graph). Since there are only 2 layers, we can fully describe a trained RBM with 3 parameters:

- The weight matrix W:
  - W has size nvisible x nhidden. Wij is the weight of the connection between visible node i and hidden node j.
- The bias vector bv:
  - bv is a vector with nvisible elements. Element i is the bias for the ith visible node.
- The bias vector bh:
  - bh is a vector with n_hidden elements. Element j is the bias for the jth hidden node.

nvisible is the number of features in the input vectors. nhidden is the size of the hidden layer.

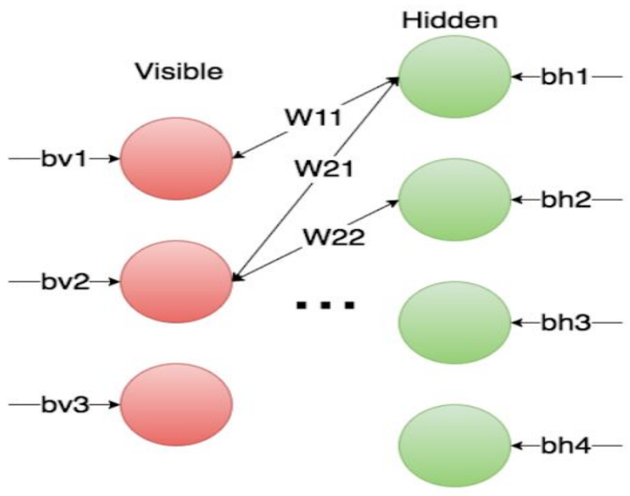


Fig 5: RBM

B. RBM Sampling

Unlike most of the neural networks, RBMs are generative models that directly model the probability distribution of data and can be used for data augmentation and reconstruction. To sample from an RBM, we perform an algorithm known as Gibbs sampling. The algorithm works like this:

- Initialize the visible nodes. They can be initialized randomly or set them equal to an input example.
- Repeat the following process for k steps, or until convergence:
  1. Propagate the values of the visible nodes forward, and then sample the new values of the hidden nodes.
     - That is, randomly set the values of each hi to be 1 with probability $\sigma(W^T v + bh)_i$.
  2. Propagate the values of the hidden nodes back to the visible nodes, and sample the new values of the visible nodes.
     - That is, randomly set the values of each v_i to be 1 with probability $\sigma(W h + bv)_i$.

At the end of the algorithm, the visible nodes will store the value of the sample.

C. RBM Training

When we train an RBM, the goal is to find the values for its parameters that maximize the likelihood of the data being drawn from that distribution.
Initialize the visible nodes with some vector x from the dataset. Sample $\bar{x}$ from the probability distribution by using Gibbs sampling and notice the difference between the x and $\bar{x}$. Move the weight matrix and bias vectors in a direction that minimizes this difference

The equations are straightforward:

$$W = W + lr(\, x^T h(x) \; - \; \bar{x}^T \, h(\bar{x})\,)$$

$$bh \; = \; bh \; + \; lr(\, h(x) - \; h(\bar{x}))$$

$$bv \; = \; bv \; + \; lr(\, x \; - \; \bar{x}\,)$$

- x_t is the initial vector
- $\bar{x}$ is the sample of x drawn from the probability distribution.
- lr is the learning rate
- h is the function that takes in the values of the visible nodes and returns a sample of the hidden nodes

This technique is known as Contrastive Divergence.

VIII. STACKED LSTM MODEL

Since music is a sequence of notes and chords, it doesn't have a fixed dimensionality. Traditional deep neural network techniques cannot be applied to generate music as they assume the inputs and targets/outputs to have fixed dimensionality and outputs to be independent of each other. It is therefore clear that a domain-independent method that learns to map sequences to sequences would be useful.

In this model, we will be using the Long Short-Term Memory (LSTM) architecture. LSTM is a type of recurrent neural network (proposed by Hochreiter and Schmidhuber, 1997) that can remember a piece of information and keep it saved for many timesteps.

A. *Model Architecture*

We will use Keras to build our model architecture. We use a character level-based architecture to train the model. So each input note in the music file is used to predict the next note in the file, i.e., each LSTM cell takes the previous layer activation ($a^{(t-1)}$) and the previous layers actual output ($y^{(t-1)}$) as input at the current time step t.

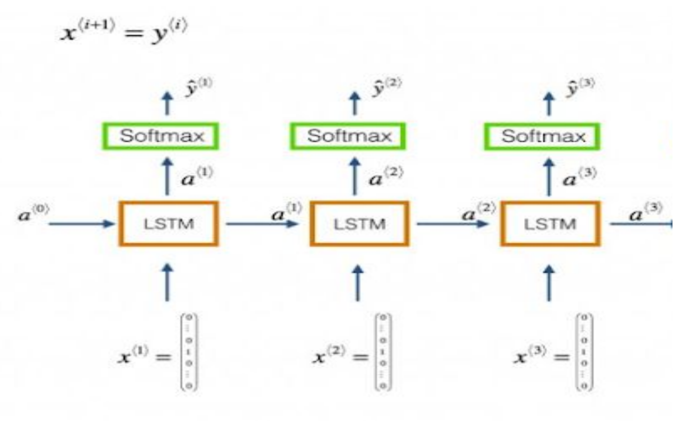
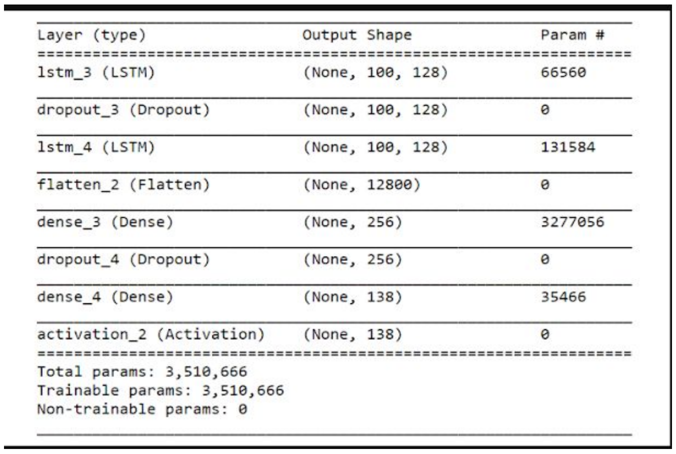


Fig 6: One to Many LSTM architecture

Our music model consists of two LSTM layers with each layer consisting of 128 hidden layers. We use '*categorical cross-entropy*' as the loss function and '*adam*' as the optimizer.



```
Layer (type)                 Output Shape              Param #
=================================================================
lstm_3 (LSTM)                (None, 100, 128)          66560
_________________________________________________________________
dropout_3 (Dropout)          (None, 100, 128)          0
_________________________________________________________________
lstm_4 (LSTM)                (None, 100, 128)          131584
_________________________________________________________________
flatten_2 (Flatten)          (None, 12800)             0
_________________________________________________________________
dense_3 (Dense)              (None, 256)               3277056
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_4 (Dense)              (None, 138)               35466
_________________________________________________________________
activation_2 (Activation)    (None, 138)               0
=================================================================
Total params: 3,510,666
Trainable params: 3,510,666
Non-trainable params: 0
_________________________________________________________________
```

Fig 7: Model summary

B. *Model Training*

The architecture gets the notes, creates the input and output sequences, creates a model, and trains the model for 200 epochs. The output of the LSTM is fed into the softmax layer over all the tokens. The loss corresponds to the cross-entropy error of our predictions at each time step compared to the actual note played at each time step.

The architecture allows the user to set various hyperparameters such as the number of layers, hidden unit size, sequence length, batch size, and learning rate. We clip our gradients to prevent our gradients from exploding. We also anneal our learning rate when we see that the rate that our training error is decreasing is slowing.

C. *Music sample generation*

To generate new notes, we need a starting note. So, we randomly pick an integer and pick a random sequence from the input sequence as a starting point.

Next, we use the trained model to predict the next 500 notes. At each time step, the output of the previous layer ($\hat{y}^{(t-1)}$) is provided as input ($x^{(t)}$) to the LSTM layer at the current time step *t*.

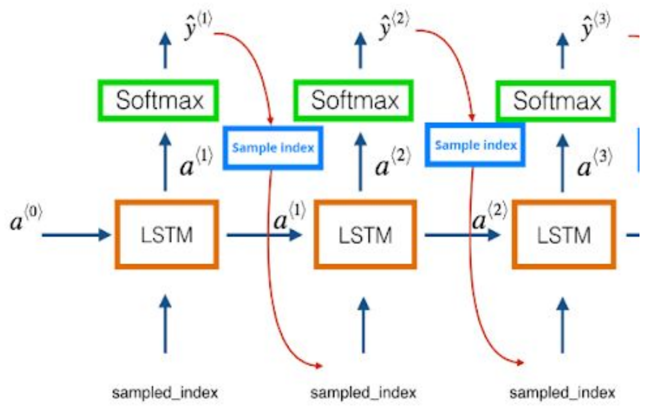


Fig 7: Sampling from a trained network

Since the predicted output is an array of probabilities, we choose the output at the index with the maximum probability. Finally, we map this index to the actual note and add this to the list of predicted output. Since the predicted output is a list of strings of notes and chords, we cannot play it. Hence, we encode the predicted output into the MIDI format and generate some new music.

IX. EVALUATION

One of the major challenges of evaluating the quality of our model is incorporating the notion of musical aesthetics. That is, how "good" is the music that our models ultimately generate? As such, there is no firm methodology to grade a music/art generative model.

One possible way is to conduct a blind experiment where volunteers or professionals in the field can offer their opinion on the samples of music generated. Therefore, there is no strict method of evaluation.

A comparison was done between the resultant audio file generated by the two employed models.

X. CONSTRAINTS, ASSUMPTIONS AND DEPENDENCIES

There are no such formal constraints or dependencies involved with the devised solutions.

Assumptions :

- Inheriting a Char RNN model for the Stacked LSTM (comparative model) model based on the assumption that music generation is similar to text generation.
- Using Gibbs Sampling as the preferred sampling method in the RBM model.

Guidelines :

Based on the analysis, the music demands long term dependencies and the generation is very similar to text generation. Therefore, based on the mentioned analysis and the research work that has gone into this field we came to the conclusion that generative models like RBM and RNN models like Stacked LSTMs would be the best candidates for the solution to this problem.

As well as, formats other than MIDI for music representation do not properly distinguish between the melody and other parts of the piece and in addition do not address the full complexity and therefore they cannot deal with the challenge of learning complex polyphonic structure in music.

XI. CONCLUSION AND FUTURE WORK

We were able to show that a generative model like RBM and a multi-layer LSTM, character-level language model applied to the data representations is capable of generating music that is comparable to sophisticated prevalent musical works. We showed that our models were able to learn meaningful musical structures.

Given the recent enthusiasm in machine learning inspired art, we hope to continue our work by introducing more complex models and data representations that effectively capture the underlying melodic structure. Another milestone is to be capable of generating music samples of the desired length. Furthermore, we feel that more work could be done in developing a better evaluation metric of the quality of a piece – only then will we be able to train models that are truly able to compose original music!

XII. REFERENCES

[1] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. Proceedings of the 29th International Conference on Machine Learning, (29), 2012.

[2] Chun-Chi J. Chen and Risto Miikkulainen. Creating melodies with evolving recurrent neural networks. Proceedings of the 2001 International Joint Conference on Neural Networks, 2001.

[3] Douglas Eck and Jurgen Schmidhuber. A first look at music composition using lstm recurrent neural networks. Technical Report No. IDSIA-07-02, 2002.

[4] Daniel Johnson. Composing music with recurrent neural networks.

[5] I-Ting Liu and Bhiksha Ramakrishnan. Bach in 2014: Music composition with a recurrent neural network. Under review as a workshop contribution at ICLR 2015, 2015.

[6] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space.

[7] Blog post introducing Magenta can be found here: http://magenta.tensorflow.org/welcome-to-magenta