The Piano Boy: An Optimized Music Generator

Wei Liu
Department of Computer Science
Georgetown University
Washington, D.C., USA
wl521@georgetown.edu

Yinzhi Xi
Department of Computer Science
Georgetown University
Washington, D.C., USA
yx157@georgetown.edu

Jie He
Department of Analytics
Georgetown University
Washington, D.C., USA
jh2063@georgetown.edu

***Abstract*—Being curious about the creativity of neural networks, we developed a neural network to generate music. In this paper, a 3-layer LSTM model is used as the basis for music development. Several improvements have been applied to increase the performance of our model successfully. We also have made some modifications to the model structure as comparisons and tested them. The final results show that our optimized model performs best among our comparative experiments and some related work.**

***Index Terms*—note, LSTM, matrix, binary cross entropy, sigmoid**

I. Introduction

Artificial intelligence is playing a much bigger role than ever. Recently, deep learning is the hottest and most exciting topic in this area. Neural networks are widely used to improve many aspects of our daily life. People used them to improve product recommendation systems, make better predictions about sports scores, generate artwork with transferred style, and so on. Since we all love music, we want to explore how neural networks can help with creating good music. It will be nice for a music fan to have some fun with a decent music generator. The goal/motivation is to improve existing AI music composers by using different neural network architectures or adding creative elements to the models. Is it possible to create an AI music composer which can produce realistic music pieces like those written by human beings? Is it possible to overcome the problem of lack of rhythm which is shared by most music generators on the market? We need solutions for them.

We chose a model with 3 LSTM layers as the foundation and made big improvements from there. One of them is to make our output music possess changing rhythm and we added some pauses to the training samples to achieve that. The other one is that we changed the inputs from single note and chord to a matrix of 88 notes (each represented by 0 or 1) to make our outputs more flexible. Besides, we also made some modifications to the model structure, tested these modified models, and used them as comparisons. Since we were building a generative model, evaluation can rely on the feedbacks of listeners. The final evaluation results show that our optimized model was the best performer among our comparative experiments and some similar applications.

II. Related Work

We conducted some research about previous and related methods which have been used to solve similar problems. McMahon, B. built AI Jukebox by using a Bidirectional LSTM architecture in Keras. Kim, J. created deepjazz, which was built on a model with 2 LSTM layers, in Keras and Theano. Svegliato, J. brought us Deep Jammer which was also built on a model with 2 LSTM layers in Theano. Sigurgeirsson, S. presented Classical Piano Composer which employed a DNN model with 3 LSTM layers. Based on our research, LSTM is commonly used in solving similar music generating problems. LSTM is considered as an extremely useful solution to deal with problems in which a network must remember information for a long period of time like music generation. LSTM can effectively represent both the long-term and short-term sequential dependencies in music. The Stacked LSTM has multiple hidden LSTM layers where each layer contains multiple memory cells and it is an extension to the original LSTM model. We picked a 3-layer stacked LSTM model as the starting point and foundation for our project.

III. Datasets

The dataset used in the project is a combination of collections of midi files, which are all classical piano music composed by famous composers like Beethoven and Mozart, from the following online data sources:

- Classical Piano Composer: 93 pieces
- Deep Jammer: 327 pieces
- Piano-midi.de: 124 pieces
- MuseData: 783 pieces
- Chopin: 98 pieces

We use a combination of the first 4 datasets as the training set, and the Chopin files as the test set. It is acceptable because these music files are all classical piano midi files with similar features. The total training set contains about 1300 pieces of midi files, and has size of around 10 MB. Each piece of midi files can generate 1000 sequences for corresponding inputs and outputs, with sequence length of 100. The test set contains 98 pieces of midi files. Each time we want to generate a piece of

music, our program choose a sequence of 100 from the test set randomly as the input and output the generated music.

A midi file is a Musical Instrument Digital Interface file and it explains what notes are played and how long or loud each note should be. For data pre-processing, we used *Music21*, a Python toolkit developed by MIT, to extract musical notation of midi files like notes and chords.

IV. Methods

A. Transforming MIDI files

We use *Music21* library, which is a open-source library for Python, to extract note and chord information from midi files.

B. Basic Structure

We use a 5 layers neural network to do this music generation task, it includes 3 LSTM layers, each LSTM layer contains 512 nodes, 1 full-connection layer with size 256 and 1 output layer. The input of the neural network is 100 beat pieces, the piece is encoded into a vector, the vector length varies in our different implementations. The output of the neural network is one beat piece. In the training stage, we generate training set from our midi files and feed it into our 5 layers model, in the generating stage, we randomly select a 100 beat piece as model input, model predicts the next beat piece and append it into input, remove the first beat piece and repeat this procedure several times.

C. Functions

In the initial implementation, we treat each metre piece as a word, basically, each metre piece is either a note or a chord, then we use one hot encoding to transform each metre to a vector, the vector dimension is the size of unique note and chord. The output dimension is also the size of unique note and chord. Since every time we need to predict a note or chord, we use *softmax* as the activation function of the output layer and use *categorical cross entropy* as the loss function.

D. Incorporating Rest Note

The initial implementation cannot output a rest note, our next implementation is to incorporate rest note, to achieve this, we read the midi file every 0.5 seconds, if there is no note or chord at that moment, we add an empty note. So the input and output vector dimension increases by one.

E. Replacing Note/Chord to Vector

Our next improvement is focused on note/chord representation, in the last two implementations, the note/chord is represented as a word (String), instead, we use an 88 length vector to represent them since piano has 88 keys. We still use *softmax* as the activation function so our model can only predict the next note and does not support multiple keys pressed at the same time.

F. Multiple Keys Prediction

The final improvement is to support multiple keys prediction. In order to generate multiple 1s in the output vector, we modify the activation function from *softmax* to *sigmoid* and use a threshold 0.5 to determine if a key is predicted as pressed or not. To work with *sigmoid* together, we use *binary cross entropy* as the loss function.

V. Results

The Stacked LSTM has multiple hidden LSTM layers where each layer contains multiple memory cells and it is an extension to the basic LSTM model. We started from a 3-layer stacked LSTM model and made some modifications to it. We employed the following variations:

- Three LSTM Layers with optimized features:
  Dimensions in 3 LSTM: (512,512,512)
  See loss figures in Fig. 1.

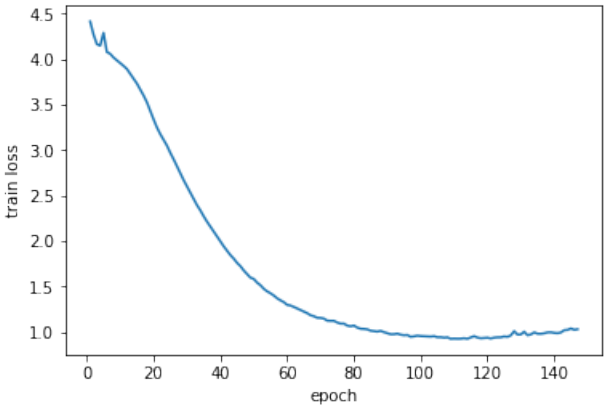

Fig. 1. Loss of 3-layer LSTM Optimized Model

- Three LSTM Layers with different dimensionality:
  Dimensions in 3 LSTM: (512,512,256)
  See loss figures in Fig. 2.

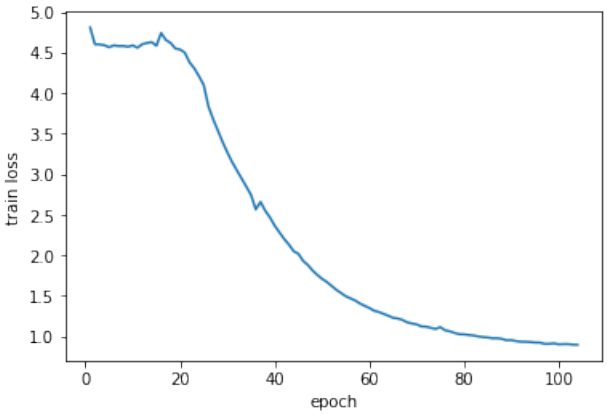

Fig. 2. Loss of 3-layer LSTM model with Modified Dimensions

- Four LSTM Layers: Add one additional hidden LSTM layer as the fourth LSTM layer.

Dimensions in 4 LSTM: (512,512,512,512)
See loss figures in Fig. 3.

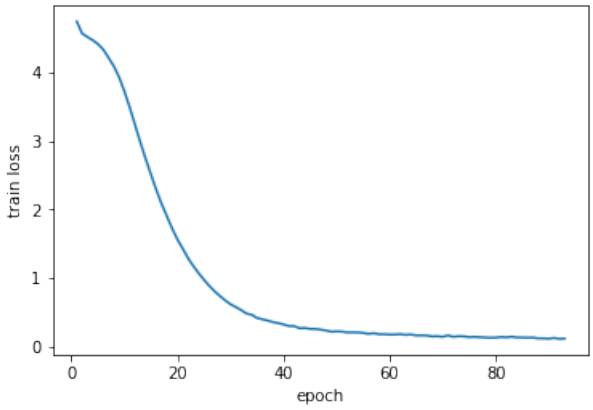

Fig. 3. Loss of 4-layer LSTM Optimized Model

It takes 20 hours to run 200 epochs with a 1080ti GPU.

After about 190 epochs, the model fits best. In order to compare the training effects with different epochs, we generated several midi files with 1, 50, 100, 152, 191 epochs separately. See comparing figures in Fig. 4.

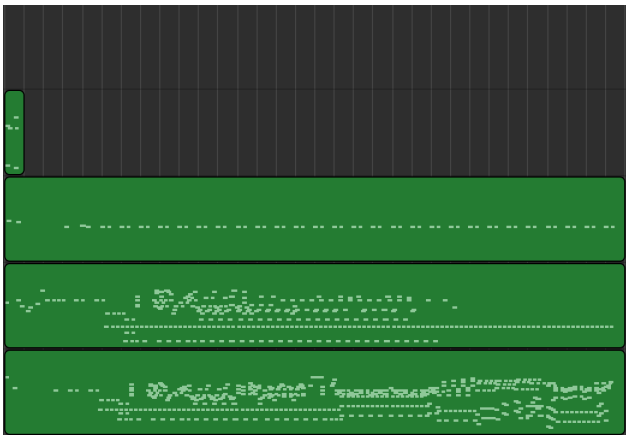

Fig. 4. Comparison results with different epochs[1]

We believe that the same 100 previous beat pieces cannot guarantee that the following keys are unique, so we don't use accuracy as the measure. Instead, we let people to listen to our generated music and music from some related work and vote for the best music. See results in Table. I.

VI. Discussion of Results

Our final model can predict rest note which output is all 0s and also can predict any combination of note and chord which output can have multiple 1s. In generated music, we can see sometimes it looks like our model can memorize the training music which is really amazing.

[1]from top to bottom are test samples with 1, 50, 100, 152, 191 epochs separately

TABLE I
Voting Results

Program Name	Number of Votes
AI Jukebox	7
Classical Piano Composer	7
Deep jazz	5
Deep Jammer	4
The Piano Boy	18

VII. Conclusions

To improve the quality of generated music, we tried several different model structure, such as three LSTM Layers model with different dimensionality and four LSTM Layers model.

We also tried to use different beat length as the input, such as use 40 instead of 100, but the result was not good. For the future directions, now our model's prediction is determined if the input is determined, but we believe that music composition is vibrant which means that same input could lead to different output, so we think a probabilistic neural network model could be the next direction.

References

[1] McMahon, B. *Music generator utilizing a Bidirectional LSTM architecture in Keras,* https://github.com/cipher813/AI_Jukebox.
[2] Sigurgeirsson, S. *Classical Piano Composer,* https://github.com/Skuldur/Classical-Piano-Composer.
[3] Justin Svegliato and Sam Witty, *Deep Jammer: A Music Generation Model*.
[4] Kim, J. *Deep learning driven jazz generation using Keras & Theano!,* https://github.com/jisungk/deepjazz.