Composing Music With Recurrent Neural Networks

E4040.2020Fall.MYEY.report
Wenjun Yang wy2347, Qihang Yang qy2231
Columbia University

Abstract

Our project aims at reconstructing the architecture of the Bi-axial LSTM model, and composing music based on the model. The code of the original paper is written in theano, while we managed to understand the practical implementation and build our model in tensorflow and keras according to the description in the original paper. After epochs of training, we manage to have some satisfying results. Our music now has a sense of chords.

1. Introduction

Mozart, Liszt, Schumann and a lot of famous composers have composed a great amount of music masterpieces with their genius talent, but models can also generate beautiful melodies as well. Music generation has long been a popular topic for researchers. Many interesting models have been proposed along the way. The main branches are deep learning models, autoregressive models[2] and traditional sequence models like the hidden Markov model, as seen in the paper by Allan and Williams[3]. As to deep learning models, RNN and its variations have been widely used in generating music, a vast amount of new models have been proposed. Among them, there are LSTM-RTRBM[4], Bidirectional LSTM[5], GRU[6], Bi-directional LSTM Model with Attention Mechanism[7] and Bi-axial LSTM [8], which all intend to address few specific features of music.

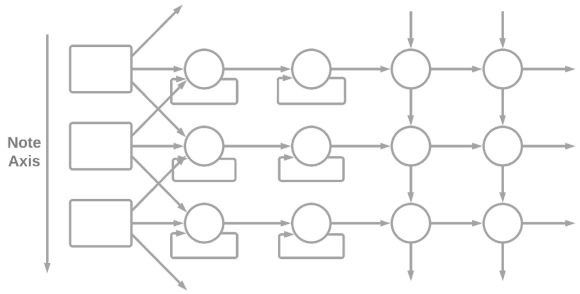
Similar to natural language, music can also be serialized as a sequence of tokens. MIDI is one of the ways to tokenize music notes in the digital era. With tokenization, we can use any sequence model to predict the next note and thus generate music. Music style has to be consistent across time, which requires the model to remember previous notes and even notes that are far away at the beginning, that is, temporal structure and long-term structure. Since LSTM specializes in coping with sequential data and is especially good at memorizing long-term dependency, it's very natural to apply it to music generation. But Music has other unique features which are hard for a vanilla LSTM to grasp, e.g, note-invariance, meaning the neural network structure should be identical for each note, and the difference between holding a note and repeating it. The original paper improves over previous work and proposes a Bi-axial LSTM model mainly to address these two problems.

The goal of our project is to reproduce the original paper with tensorflow and keras. The biggest challenge for us is to grasp the idea of the model in the original paper, for it combines data science and music domain knowledge and writes the whole architecture in theano. Luckily, we have the time-distributed layer in keras, which perfectly mimics the author's implementation in the paper. To better understand the logic behind the model, we also take some time to get familiar with MIDI notations, which better prepares us for this project.

2. Summary of the Original Paper

2.1 Methodology of the Original Paper

The Bi-axial model proposed by the original paper stacks LSTM first in time axis then in note axis. The first two layers are independent in notes, but have connection across time. The last two layers are connected in the note axis while independent across time. Its structure is displayed in the below picture, note the loop means delay by one time step.



To understand the input data, we need to first introduce the notion of state matrix. Since MIDI uses NoteOnEvent and NoteOffEvent to denote the start and end of a note, the author translates these events into a matrix of shape (# time step, # note, 2 states). The first state denotes whether the note is played last time step, and the others denote whether it is articulated last time step. In digit, this rule can be interpreted as the following:

(1,1) the note was played last timestep and is articulated in last timestep

(1,0) the note was played last timestep but is not articulated in last timestep

(0,0) the note wasn't played last timestep

The input data has been carefully devised to keep the most of information so as to provide information for prediction. The input data of the the whole model is of the

shape (#time step, #note, 80). The 80 features of each note and each time step consist of the following:
[0] note: the figure denoting note in MIDI track, will be subtracted by the lower bound of notes we are interested in.
[1:13] pitch: one-hot encoded vector (length 12) denoting which pitchclass the note belongs to, e.g. CDEFGABC#.
[13:63] previous vicinity: states for the surrounding octave in each direction, 2 for one note, one for play, the other for articulate, then encode it as a vector of size 50
[63:75] context: get the count of pitch class played in the last timestep, then encode it as a vector of size 12
[75:79] beat: beat of music
[79] 0: filler

The numbers of hidden states chosen by the original paper is 300, 300, 100, 50 respectively for each LSTM layer.

Note at the end of time-axis layer (2nd layer), the original paper adds extra input data to the model, i.e., whether the previous note (half-step lower) is played last time step, making the input shape of note-axis layers to be (#time step, #note, previous output dimension+2).

The output of the model is the probability of playing or articulating the note next time step (1, 78, 2). The loss is calculated by comparing the output to the ground truth, that is, the next note matrix of the time step after the end of our input sample.

The generation process is predicting the next time step, appending the result to the original sample and saving the result to the output matrix, then moving the window one time step further. Repeat the process and transform the output into a MIDI track, that way we can get the generated music.

2.2 Key Results of the Original Paper

The original paper proposes a neural network architecture (Bi-Axial LSTM) that can preserve the translation-invariance of the music. Bi-Axial LSTM divides the music generation and prediction task such that each network instance is responsible for a single note and receives input relative to that note. Moreover, Bi-Axial LSTM performs better on prediction tasks than similar non-parallel models. However, current Bi-Axial LSTM can only model measure-level structure and can not preserve long-term structure.

3. Methodology (of the Students' Project)

Our methodology generally follows a similar pattern with the original paper, but also differs in several ways. To start with, our project uses tensorflow and keras to implement the paper while the original paper is based on theano. Also, with limited time and computing resources, we only manage to train the model for 1050 epochs in contrast to 10000 epochs in the original paper, that might cause huge differences in the output. To see the exact architecture we are using and details of training, please refer to section 4.1.

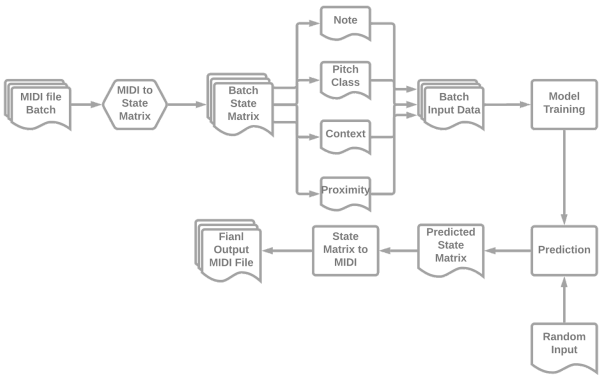
3.1. Objectives and Technical Challenges

Our objective is to use Bi-axial LSTM to generate an original music piece from a list of classic piano midi files. The technical challenges can be concluded as these steps:

- Change the midi files into data and state matrics. We need to derive the 80 features of midi files, which has been mentioned above, and transfer origin midi files into matrices with shape (timesteps, notes, features). Then, transfer them into state matrics with shape (timesteps, notes, 2 states). This is the ideal input of our model.
- Create our Bi-axial LSTM model using Tensorflow. In the original paper, they use theano to create their model on python 2.0, which is a little out of date. We need to understand the model structure and build it on Tensorflow using similar functions.
- Process the prediction and generate original music pieces. After training our models, we need to generate a slide of predicted data, and transfer them into midi files, which can be played by a player. This process includes an automatic generation procedure and a transformation from data to midi procedure.

3.2.Problem Formulation and Design Description

The general workflow of the project is as follows.



In other words, we train our model written in tensorflow and keras, predict and generate music based on our model, then compare our output music with the products of the original paper.

The pseudo code of our procedure can be described as:

Input: A list of midi files, the length of generated music

Output: A piece of original music	
1	Load the midi files and transfer them into data type.
2	Transfer the data type into state matrices and concatenate them as training data.
3	Input the training data into our Bi-axial LSTM model, compile and train it.
4	Randomly select an example in the training data, and use the model to predict it.
5	Concatenate the prediction on the example, delete its first time step and use is as a new test data.
6	Repeat step 5 until the prediction reaches the desired length.
7	Transfer the prediction into a midi file and output the music.

4. Implementation

In this section, we will talk about how to implement our algorithm and the figure of parameters we used in our programming.

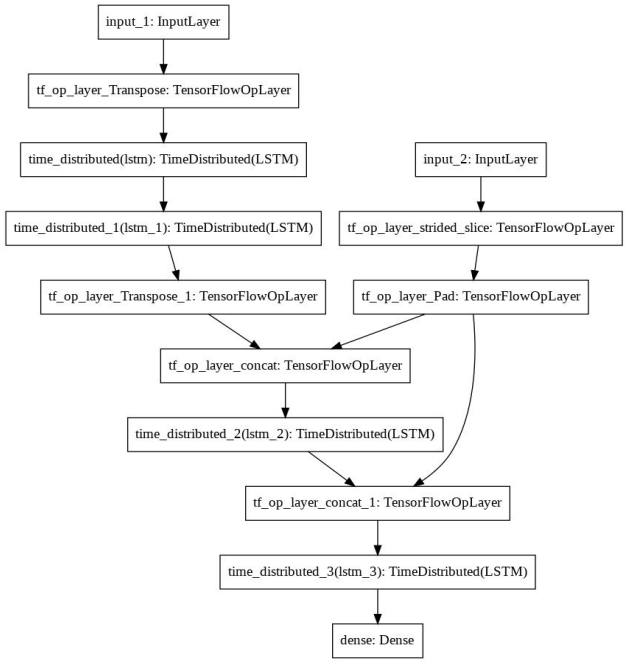
4.1. Deep Learning Network

The dataset we are using is identical to the original paper, which can be found in Classical Piano MIDI Page [9]. They are a list of classical piano midi files. We select 127 time steps as the length of a piece of training data.

The original paper implements the model in theano, and we use tensorflow and keras instead. The following picture describes the architecture of our model.

Our model involves 2 transpose layers in order to change the structure of input data into the form that caters to the time-distributed layers. Also note even though we have two separate layers in the diagram, in practice, we combine these 2 together and form a single input layer.

We train our model for 600 epochs with 20 steps for each epoch and 10 samples for each step, each sample is of length 128 time steps according to the original paper. We use Adam optimizer with a learning rate of 5e-5 and a custom loss function calculating the negative log likelihood of our output against the true state matrix of note being played next time step.



A standard LSTM has an input gate, a forget state and an output gate. The formulas can be presented as:

$$
\begin{aligned}
i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \\
f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \\
c_t &= i_t \odot z_t + f_t \odot c_{t-1} \\
o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \\
h_t &= o_t \odot \tanh(c_t) \\
z_t &= \tanh(W_{xz}x_t + W_{hz}h_{t-1} + b_z)
\end{aligned}
$$

Because that the prediction is probability, we create our loss function by modifying on the binary entropy:

$$C = -\sum_{t=1}^{T}\sum_{n=1}^{N} \ln[v_n^{(t)} q^{(n,t)} + (1 - v_n^{(t)})(1 - q^{(n,t)})]$$

where $q^{(n,t)}$ represent the prediction of $p^{(t)}(v_n=1|v_{<n})$.

To be more detailed, the working procedure of our model can be described as:

1. We choose 127 timesteps as the length of our training music, so in our Bi-axial LSTM model, supposed that one piece of input data is X, it will have a shape of (127, 78, 82). We can divide this matrix into a data matrix with 80 features and a state matrix with 2 states. Suppose that $X = (X_d, Xs)$, where the X_d matrix has shape (127, 78, 80) and the X_s matrix has shape (127, 78, 80).
2. The X_d input will go through a transpose layer and its shape will be changed into (78, 127, 80), and then go into two time distributed LSTM each with 300 units, which means to calculate a LSTM on each 127 timesteps, and the whole output shape is (78, 127, 300). Then X_d will be

transformed back into (127, 78, 300) and wait for later action.

3. In the meantime, the X_s input will make a displacement, to delete the last timestep and add a zero padding at the beginning timestep, and then this state matrix will be concatenated to the processed X_d, to create a large matrix with shape (127, 78, 302).
4. The large matrix will then go through a LSTM with 100 units and obtain a (127, 78, 100) output. We concatenate the state matrix to the output again, and make them go through a LSTM with 50 units together, and the output will have shape (127, 78, 50).
5. The last part of our model is a flatten layer, a dropout layer with 0.5, and a dense with 156 units and activated by sigmoid function, the final output will be reshaped into (78, 2) shape, which means the probability of state matrix to be positive of the notes in next time step after training data.

After the model is built and trained, our next step is to use the prediction of our model to generate original music pieces. We randomly select a music piece in our training data and use its first 127 timesteps as a starting data to make a prediction. The prediction is the probability of the state matrix to be positive of the note at time step 128. We create a uniformly random variable and compare it with the probability to derive an actual state matrix of this note. We transfer the state matrix to the input data type and concatenate it to our starting data, and select the last 127 timesteps as our new training data. Keep it on and we can continually generate a long list of data. Choose the ideal length of the data matrix and transfer it back to a midi file, we finally generate an original music piece.

4.2. Software Design

The flowchart and pseudo code of our whole procedure is mentioned in 3.2, and here we want to describe some key steps in the flowchart in detail.

- Midi to State Matrix

Input: A list of midi files
Output: Each file's state matrix

```
global_time = 0
TPQN = ticks per quarter note
While True Do
  If global_time % (TPQN/4) == （TPQN/8) Then
      Switch to a new state
      new_state = [old_state[0], 0]
  For i in number of tracks Do
      If current track's current event is midi.NoteOffEvent or velocity == 0 Then
          current state is [0, 0]
          cursor of current track moves to next event on current track
      If current track's current event is midi.NoteEvent Then
          current state is [1, 1]
          cursor of current track moves to next event on current track
      If the music sample changes its time signature in the middle Then
          stop the transformation
      If we have observed all the event in all the tracks Then
          break
global_ time = global_time + 1
```

- Generate Music

Input: A midi file as starting_data, length of the generated music, name to save
Output: A generated midi music

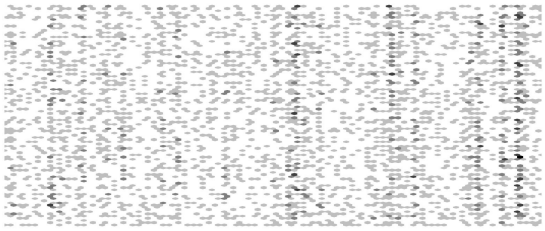
```
Select a part of the midi file as starting_data
For i in length of the generated music Do
   pred_state = model.predict(starting_data)
   For j =1 to 78 Do
      prob = random number in (0,1)
      If pred_state[1] > prob Then
          pred_state[1] = 1
      Else
          pred_state[1] = 0
      prob2 = random number in (0, 1)
          pred_state[2] = pred_state[1] * (prob2 < pred_state[2]
      transfer pred_state to data matrix and concatenate it to starting_data
   delete the first time step in starting data and continue
```

5. Results

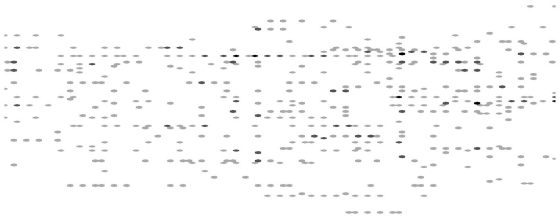
5.1. Project Results

We generate a few pieces of music and select some of our best products for display. Because of the limitation of the pdf format, we decide to visualize the music we generate. Each dot of the following pictures denotes a note. From top to down, the vertical axis represents the note from 128 to 1, while the horizontal axis denotes the time steps. However, it doesn't show how long a note will be held.

The first picture below is the music generated by our model with random initialization, the following three are results generated after 1050 epochs. Please refer to the samples folder in our github[1] for a few clips to listen to.



(Random Initialization)



(Epoch 1050 Sample 1)

(Epoch 1050 Sample 2)

(Epoch 1050 Sample 3)

5.2. Comparison of the Results Between the Original Paper and Students' Project

We also bring up one piece of music from Issac Albéniz and one prediction from the original paper for comparison.

(Issac Albéniz)

(Sample from Original Paper)

Compared with a random initialization model, our model does learn a lot, and has a primitive sense of chord. Notice in sample 1, 2, 3, many notes come in pairs, which could be deemed as a prototype of chords. But still it's far from the harmony as in the work of Issa Albéniz. Some of the generated samples still sound haphazard and chaotic. Also, it doesn't learn how to hold a note, so every note in the sample is rather short, which will be the one of our focus in future work. In general, our model gives pretty stunning results after merely 1050 epochs of training, compared with 10000 epochs in the original paper.

5.3. Discussion of Insights Gained

Our implementation differs with the original paper in the following ways:

- First is the difference of deep learning framework. The original paper used theano to create their model, but theano is a little outdated at present. We changed to use tensorflow, which is more familiar to us. The implementation of the deep learning model also needs to be adjusted, but the structures of the model are similar.
- We used different optimizers to train our model. The original paper used RMSprop with a learning rate of 0.001 as their optimizer , but we

chose Adam with a learning rate of 5e-5 to increase the training spend and accuracy.

- The original paper used the average of the binary cross entropy as the loss function. As for making the loss more obvious, we used the sum of binary cross entropy as our cost, and this change causes little difference.
- As to the training epochs, because that the model training takes a long time, we did not train as many times as the original paper, and used 1000 epochs as the number of iterations, and we think the output is meaningful enough.

6. Conclusion

In this project, we reconstructed the Bi-axial LSTM model in tensorflow and Keras. After epochs of training, we predict the next note timestep by timestep, and eventually generating pieces of music which have a sense of chords.

We learned many lessons in this project:

- A key point in the original paper is that it uses state matrix to represent midi files cleverly, which makes the prediction by deep learning model more feasible. How to deal with complicated data is the key issue in this kind of problem. The displacement of state matrices in training data also makes the LSTM more meaningful, which is a creative idea.
- A proper deep learning model should take more data properties into consideration, and this Bi-axial model can take care of both the input data and its state matrix, along with the time sequence, improving the performance of the original LSTM model.
- The number of training epochs really make a significant difference in a large deep learning model, and it is easy for the training time to exceed several hours. Also we have trained our model for 1000 epochs, the output still can be improved.

Many potential improvements have been left for the future due to lack of time. Future work will involve deeper analysis of current mechanisms and also trying different methods.

- In particular, due to lack of time and computing resources, we only managed to train our model for 1050 epochs. Thus, priority will be given to further training.
- Moreover, velocity in the current model is fixed at a hand-pick value, which is partly due to the nature of the current dataset. However, music with only one velocity would be very bland, skilled pianists often have their own interpretation of the music and emotions, which results in the different velocity and style even in the same note of the same song. It would be nice to cover some performance or style in the model. Many researchers have carried out similar research, many of them use datasets of MIDI music of skilled pianists and some of them [10] successfully predict the notes and also their expressive timing and dynamics.
- Bidirectional LSTM is also a promising field to look at. Generally, Bidirectional LSTM provides a more thorough view of the entire sequence, which would be nice to be applied to music generation, because music usually is a sequence of notes that follow the same style and interdependent across time. Plus, Bidirectional LSTM provides faster and even fuller learning of the problem.

7. Acknowledgement

We would like to express our deep gratitude to Professor Zoran Kostic for the wonderful journey in the deep learning world this semester. We would also like to thank all of our TAs, for their advice and hard work. Our grateful thanks are also extended to the developers of tensorflow and keras for their hard work to make deep learning a much easier task for beginners. Also, big thanks to all the researchers and tech bloggers for sharing their thoughts and insights on deep learning, especially the LSTM model.

8. References

[1] Github

[2] S. Dieleman, A. v. d. Oord, and K. Simonyan, “The challenge of realistic music generation: modelling raw audio at scale,” *Advances in Neural Information Processing Systems*, pp. 8000–8010, 2018.

[3] M. Allan, and C. K. Williams, “ Harmonising chorales by probabilistic inference,” *Advances in Neural Information Processing Systems*, 17:25–32, 2005.

[4] Q. Lyu, Z. Wu, J. Zhu, and H. Meng, Modelling high-dimensional sequences with lstm-rtrbm: application to polyphonic music generation, *Proceedings of the 24th International Conference on Artificial Intelligence*, pp. 4138–4139. AAAI Press, 2015.

[5] T. Jiang, Q. Xiao and X. Yin, "Music Generation Using Bidirectional Recurrent Network," *2019 IEEE 2nd International Conference on Electronics Technology (ICET)*, Chengdu, China, 2019, pp. 564-569, doi: 10.1109/ELTECH.2019.8839399.

[6] A. Nayebi, and M. Vitelli, “GRUV: Algorithmic music generation using recurrent neural networks,” *Course*

CS224D: Deep Learning for Natural Language Processing (Stanford), 2015.
[7] A. Ranjan, V. Behera, M. Reza, Using a Bi-directional LSTM Model with Attention Mechanism trained on MIDI Data for Generating Unique Music, arXiv e-prints, 2020.
[8] D.D. Johnson, "Generating Polyphonic Music Using Tied Parallel Networks," *Correia J., Ciesielski V., Liapis A. (eds) Computational Intelligence in Music, Sound, Art and Design(EvoMUSART), Lecture Notes in Computer Science*, vol 10198. Springer, Cham, 2017. .
[9] Classical Piano Midi, Classical Piano Midi Page, 1996-2018 by Bernd Krueger. [Online]. Available: http://www.piano-midi.de/midi_files.htm
[10] Oore, S. et al. "This time with feeling: learning expressive musical performance." *Neural Computing and Applications* (2018): 1-13.

9. Appendix

9.1 Individual Student Contributions in Fractions

	UNI1	UNI2
Last Name	Yang (wy2346)	Yang (qy2231)
Fraction of (useful) total contribution	2/3	1/3
What I did 1	Write the data preprocessing part	Write the predict and compose functions
What I did 2	Write the model and visualization	Write the algorithms part
What I did 3	Write part of the report	Write part of the report