CS230

Music Box for CS230-Winter 2018

Somnath Purkayastha
Department of Computer Science
Stanford University
psomnath@stanford.edu

Abstract

In the year 2016, Google brain team (Magenta) released a simple 90 seconds tune generated by the trained neural network, and research on AI generated music exploded. People used many creative ways to collect and preprocess input data to train deep network to produce more and more complex pieces of music. The purpose of this project was to produce a neural network model, Music Box, which can be trained with classical piano music and can produce equivalent but original music pieces.

1 Introduction

Since the beginning of the human civilization, artists produced music to entertain. Around 2000 BC, even before parchment or paper were used for writing, they also figured out a way to record music in stone using musical notations. Looking into those music sheets, people always wondered, if music can be generated automatically. In recent years (in 50's) people tried to find statistical pattern in classical music to mimic, with exciting but limited success.

Deep learning, given enough data and processing power, can learn very complex data representation. Especially a specific deep learning architecture, Recurrent Neural Network (RNN), is extremely good to recognize the pattern in a complex time series data. So, it is no wonder, people tried to use RNN model to learn and generate music sequence, with success.

In this project I have used a Long Short-term Memory (LSTM) model, a type of RNN, which was trained with preprocessed music notations. After training for 70+ epochs (cycles), the same model was used to generate an original music piece.

2 Related work

One of the seminal paper on RNN[5], Andrej Karpathy shows us the power of RNN to generate sequence. Chun-Chi J. Chen & Risto Miikkulainen in their paper [1] showed how to use reinforcement learning, instead of explicit composition rules, to generate melodies. Though RNN was a powerful solution to generate music, due to gradient problem it was not suitable to understand the global structure of a music composition. Douglas Eck & Jurgen Schmidhuber in their paper [2] introduced the concept of using LSTM to generate music composition. In 2014, I-Ting Liu & Bhiksha Ramakrishnan [4] used resilient propagation (RProp), instead of the standard back propagation through time, with LSTM to achieve much better result generating quality music. Next Allen Huang & Raymond Wu used their RNN based model [6] to generate music with not only harmony, but also with melody.

3 Dataset and Features

I wanted to train the LSTM model with music data, more specifically Piano music data. The MIDI music format was very suitable to extract music notations, hence I looked for a good dataset of Piano music files in Midi format. E-piano competition website[15] had a rich collection of files from different piano competitions over multiple years. I used a popular python package, beautifulsoup[12], to scrap the website and downloaded more than 2500 high quality MIDI files with piano music.

My next step was to extract music notations (example given below) from those files using another popular python package, Music21[13]. After extraction, the data looked some like below.

['Rest', 'Rest', 'Rest', 'Rest', 'Rest', 'Rest', 'D4', 'Rest', 'Rest', '2.3', 'D4', '2.3.6', 'D4', '5.6.9', 'D4', '2.3.6',.................. '2.4', '2.6.9', 'A7', '5.7', 'Rest', 'Rest']

I applied three preprocessing steps, before feeding the data in my model – codification, normalization and vectorization.

3.1 Codification

Every unique note or chord (element) in the extracted data was replaced by corresponding unique integer number. I maintained a dictionary of all unique elements. After this processing steps I had 425 unique elements, observed in the data, in my dictionary.

{'Rest': 0, 'G4': 1, '0': 2, '4.7': 3, '0.4': 4, 'G4': 5, 'F4': 6, '2.5': 7, 'G3': 8, '11.2': 9, 'G3': 10, 'D4': 11, '0.3': 12,............... '2.3.6': 423, '11.0.2.4.5': 424}

After this codification step, the extracted music notation data looked like a list of integers.

[0, 0, 0, 0, 0, 0, 11, 0, 0, 158, 11, 423, 11, 425, 11, 423, 11, 77, 426, 427, 11, 423, 11, 423, 11, 425, 11, 423, 11,.............. 110, 148, 449, 73, 0, 0]

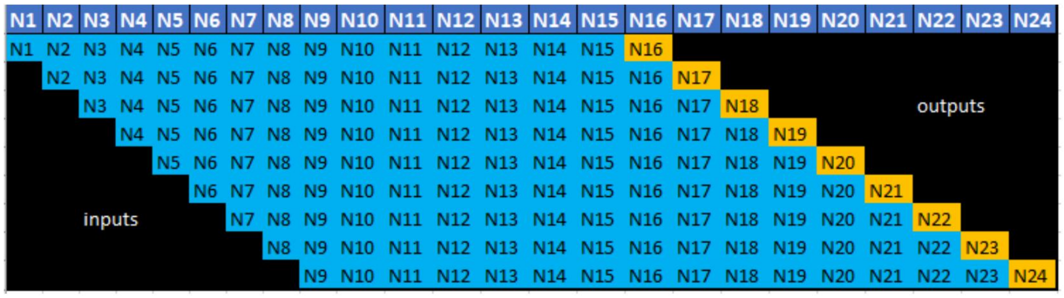
3.2 Normalization

To improve the computation efficiency, all those integers in the previous steps were divided by 425 (number of unique elements in the dictionary), to get a normalized list.

[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.023655913978494623, 0.0, 0.0, 0.33978494623655914, 0.023655913978494623,.................................., 0.23655913978494625, 0.31827956989247314, 0.9655913978494624, 0.15698924731182795, 0.0, 0.0]

3.3 Vectorization

To train the LSTM model, I needed to provide many of example of input/output data, where the input was a list of normalized values for elements and the output was the corresponding next expected element. To prepare this dataset I used a sliding window approach. In this mock example we assumed the number of elements we have received from the previous step was 24, and the length of each input sample is 15.


N1 N2 N3 N4 N5 N6 N7 N8 N9 N10 N11 N12 N13 N14 N15 N16 N17 N18 N19 N20 N21 N22 N23 N24
N1 N2 N3 N4 N5 N6 N7 N8 N9 N10 N11 N12 N13 N14 N15 N16
N2 N3 N4 N5 N6 N7 N8 N9 N10 N11 N12 N13 N14 N15 N16 N17
N3 N4 N5 N6 N7 N8 N9 N10 N11 N12 N13 N14 N15 N16 N17 N18
outputs
N4 N5 N6 N7 N8 N9 N10 N11 N12 N13 N14 N15 N16 N17 N18 N19
N5 N6 N7 N8 N9 N10 N11 N12 N13 N14 N15 N16 N17 N18 N19 N20
N6 N7 N8 N9 N10 N11 N12 N13 N14 N15 N16 N17 N18 N19 N20 N21
inputs
N7 N8 N9 N10 N11 N12 N13 N14 N15 N16 N17 N18 N19 N20 N21 N22
N8 N9 N10 N11 N12 N13 N14 N15 N16 N17 N18 N19 N20 N21 N22 N23
N9 N10 N11 N12 N13 N14 N15 N16 N17 N18 N19 N20 N21 N22 N23 N24

Finally each element of the output list was converted to one-hot vector, with 425x1 dimension.

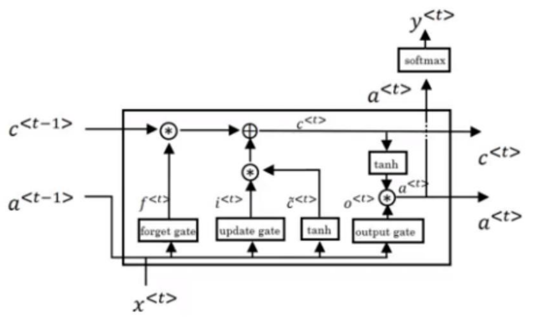
After training the model was used to generate a list of notes with an initial seed input (one of the random vectors from the training input). The raw output from the model was a list of one-hot vectors, which went through a postprocessing step to be converted into a music (MIDI) file[17].

4 Methods

In deep learning, vanilla RNN (Recurrent Neural Network) is an excellent choice to understand the data representation in a time series, but it is very difficult to train due to vanishing/exploding gradient, especially when the time series is long (have to remember a lot of history). LSTM is an advance variation of RNN, where the network uses gates to decide what to forget, what to update and what to output [16].

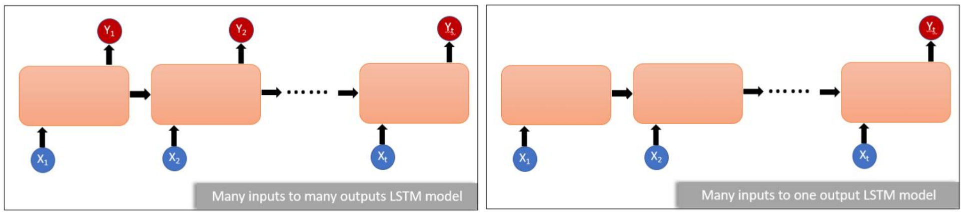
LSTM in pictures

$$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$
$$\Gamma_u = \sigma(W_u[\, a^{<t-1>}, x^{<t>}] + b_u)$$
$$\Gamma_f = \sigma(W_f[\, a^{<t-1>}, x^{<t>}] + b_f)$$
$$\Gamma_o = \sigma(W_o[\, a^{<t-1>}, x^{<t>}] + b_o)$$
$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$
$$a^{<t>} = \Gamma_o * \tanh c^{<t>}$$



I used the open source neural network library, Keras [11] with TensorFlow [10] backend, to implement my model. The model had three initial LSTM layers followed by two fully connected layers. Finally, the result went through a SoftMax layer to generate the predicted note.

The first two LSTM layers were configured to have many inputs and many outputs, where the third LSTM layer was configured to have many inputs but only single output.


Y_1
Y_2
Y_t
X_1
X_2
X_t
Many inputs to many outputs LSTM model
Y_t
X_1
X_2
X_t
Many inputs to one output LSTM model

To learn better features, regularization technique was used by implementing dropout layers in the model.

In every iteration the model predicted one of the 425 possible values (music elements in our dictionary) as output, hence we have used categorical cross entropy as our loss function :

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

[where M - number of classes, y – binary indicator if class label c is the correct classification for observation o, p – predicted probability observation o is of class c].

5 Experiments/Results/Discussion

To train the model I used (and tuned) a number of following parameters and hyperparameters.

5.1 Number of Notes:

This value determined, how many notes were used as input data for the preprocessing step. The higher value resulted in better model, but it took longer to preprocess the data and train the model. My final selection was 50,000 notes.

5.2 Number of epochs:

This value controlled the number of iterations to train the model. The higher value resulted in better model (with lower error rate), but I ran out of time for this project after 76 epochs.

5.3 Size of LSTM layer input data:

This value determined the number of input neurons for the LSTM layer. Though I had three different LSTM layers, I kept this parameter same for all those layers to 512 with good outcome.

5.4 Size of dense layer input data:

This value determined the number of input neurons for the fully connected layer. Though I had two different Dense layers, I kept this parameter same for all those layers to 256 with good outcome.

5.5 Sample Size:

During the preprocessing step, this number determined the sliding window size. The higher number required more computation power. I kept this number to 100.

5.6 Batch Size:

To improve the training efficiency, I used batch to train the model. My final setup for the batch size was 64.

5.7 Dropout Rate:

For regularization I used dropout layers in between each LSTM and fully connected layers. After trying different values between 0.1 and 0.8, I found the best result (lowest error, reasonable time to converge) with 0.3 value.

5.8 Activation Function:

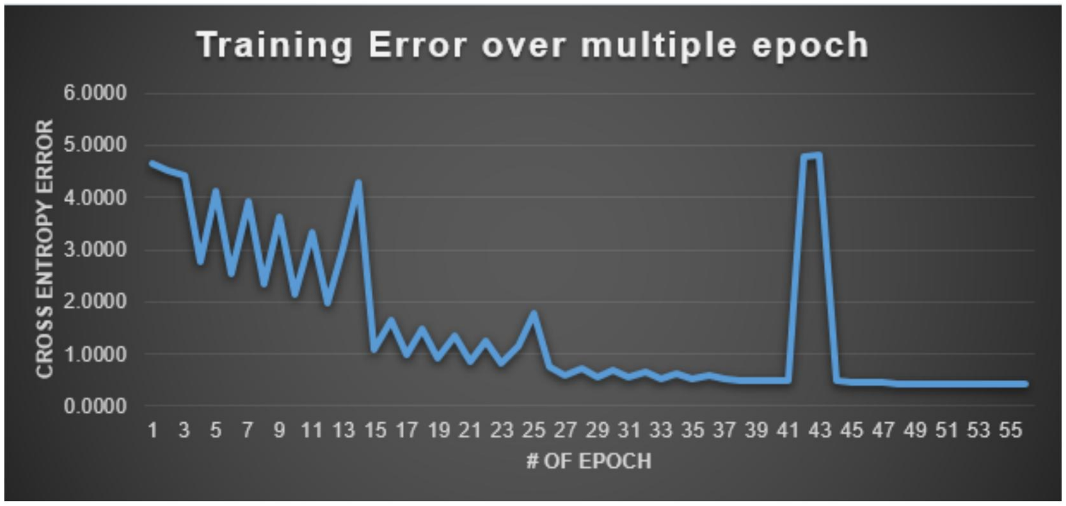
Considering the output requirement, I only considered ‘Softmax’ as my activation function, which gave the highest probability for the predicted class.

5.9 Loss Function:

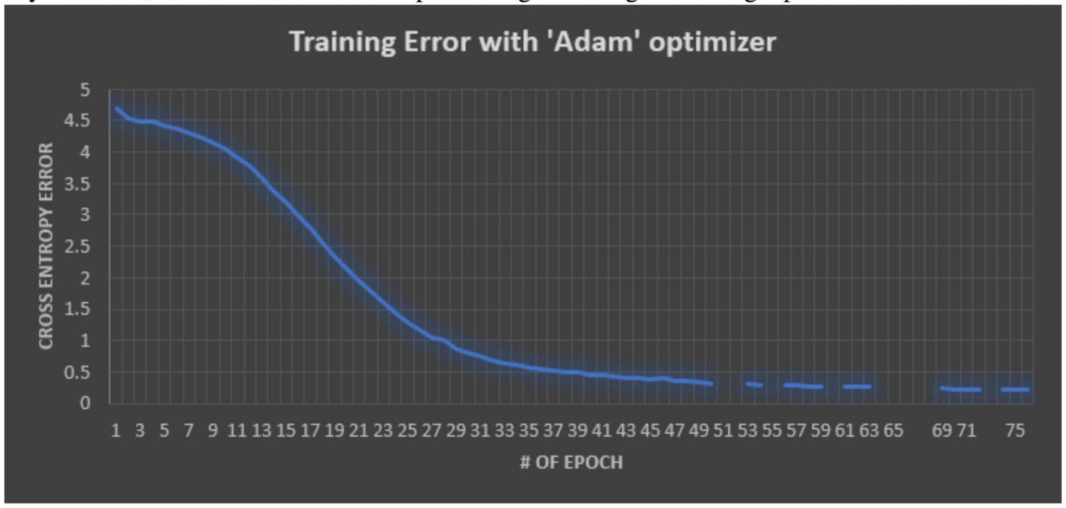
As mentioned above, I used ‘Categorical Cross Entropy’ loss function to determine if the correct class out of 425 possible options.

5.10 Optimizing Function:

During the training phase I tried several different optimizers (’sgd’ ,’rmsprop’ ,’adagrad’ ,’adadelta’ ,’adam’ ,’adamax’ ,’nadam’) to converge the data. My initial favourite was ‘rmsprop’, which reduced the error very quickly, the reduction was not a smooth curve. Also, after (41st and) 55th epoch the error went up again.



Next, I tried the 'adam' optimizer with learning rate decay, to avoid the fluctuation I observed earlier. Due to my limited training time and resource, I was unable to find the optimal value for the decay rate. Fortunately, after removing the learning rate decay function, the standard 'adam' optimizer gave me good enough performance to continue.



[Note: The missing parts of the graph represent those epochs when the error didn't improve.]

6 Conclusion/Future Work

In this paper, an RNN based LSTM model was proposed, which was trained to learn the musical structure from a list of input music files (in midi format) and was used to compose a new music piece. During the training phase, the prediction error went down with increased number of epochs.

Piano music is more than playing the specific note or chord at the right time. Maestros often use a technique called velocity (how rapidly or forcefully the key on a piano is pressed), to render a master piece. Though my model was successful to generate an original and interesting piece (a list of notes and chords in a time series), I felt it was missing its soul. In future I will plan to include Velocity in my input for the model.

After the first iteration (epoch), the model was only generating music repeating a random single note. With more iterations, the quality improved significantly. I will plan to train the model for longer period of time and with more computational power to get better result (less prediction error).

Finally, I will plan to expand the model to generate music for other instruments (like Tabla) and eventually multiple instruments (for orchestra) at the same time.

7 Contributions

This work was the final project for the ‘CS230 Deep Learning’ course offered by Stanford University. I want to thank the course TA, Abhijeet Shenoi, for his valuable inputs and guidance.

I am also extremely thankful to Sigurður Skúli, whose blog post[8] was the cornerstone to develop the model described in this paper.

References

[1] Chun-Chi J. Chen & Risto Miikkulainen (2001). Creating Melodies with Evolving Recurrent Neural Networks.

[2] Douglas Eck & Jurgen Schmidhuber(2002). A First Look at Music Composition using LSTM Recurrent Neural Networks.

[3] Nicolas Boulanger-Lewandowski, Yoshua Bengio & Pascal Vincent (2012). Modeling Temporal Dependencies in High-Dimensional Sequences: Application to Polyphonic Music Generation and Transcription.

[4] I-Ting Liu & Bhiksha Ramakrishnan (2014). BACH IN 2014: MUSIC COMPOSITION WITH RECURRENT NEURAL NETWORK.

[5] Andrej Karpathy (2015). The Unreasonable Effectiveness of Recurrent Neural Networks.

[6] Allen Huang & Raymond Wu (2016). Deep Learning for Music.

[7] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior & Koray Kavukcuoglu (2016). WAVENET: A GENERATIVE MODEL FOR RAW AUDIO.

[8] Sigurður Skúli. How to Generate Music using a LSTM Neural Network in Keras. https://towardsdatascience.com/how-to-generate-music-using-a-lstm-neural-network-in-keras-68786834d4c5

[9] Gaurav Trivedi. Machines learn to play Tabla. https://www.trivedigaurav.com/blog/machines-learn-to-play-tabla/

[10] Tensorflow https://www.tensorflow.org/

[11] Keras https://keras.io/b

[12] Beautiful Soup https://www.crummy.com/software/BeautifulSoup/

[13] Music21 : a toolkit for computer-aided musicology http://web.mit.edu/music21/

[14] Magenta – Make Music and Art Using Machine Learning https://magenta.tensorflow.org/

[15] Piano-e-Competition - http://www.piano-e-competition.com

[16] Coursera Course : Deep Learning (course 5)

[17] Sample music generated by the model, described in this paper - https://soundcloud.com/user-856463695/76-weight-loss-0-2099/s-ZzcMO

[18] iPython notebook with project code (MusicBox.ipynb)- https://notebooks.azure.com/SomnathPurkayastha/projects/CS230