Recurrent Neural Networks in Musical Composition

E4040 Fall 2017 Final Project Report

Bangda Sun (bs2996), Kai Chen (kc3041), and Sean Reddy (sr3336)
Columbia University

Abstract

Neural networks serve as a framework through which we can gain insight and solve particularly complex problems, some which are even difficult for humans. In this paper we explore the adaptation of recurrent neural networks (RNNs) to see if we are able compose music similar to that produced by humans. Using Daniel Johnson's Theano implementation in his paper entitled "Composing Music with Recurrent Neural Networks" [2] as a starting point, we sought out to port this framework into Tensorflow in Python 2 with our primary goal: to create a pipeline for music composition. Ultimately, we found that we were able to build and train a model to learn to generate music using this framework.

1. Introduction

A common theme that arises in the world of artificial intelligence is the attempt to replicate the ability to do something that a human is capable of. The area of musical composition is a notably challenging field of the arts to replicate. However, with the recent surge in computational power, and a growing understanding for neural networks in general, we sought out to see if we are able to create a network that is capable of solving this problem.

We utilized Daniel Johnson's research, a Theano implementation of RNNs in music composition[2], as a starting point and continual reference throughout our own research. With impressive findings, we sought out to replicate his results using our interpretation of his ideas presented in this paper. The majority of the effort was placed in converting code into Tensorflow and ensuring that we retain the proper network structure and approach. We found that we were able to replicate this pipeline fairly accurately, however our biggest comparative shortcoming was the relative resources available to us in the form of both time and computing power. Despite these challenges, we were able to implement an LSTM-based RNN that is capable of musical composition, albeit with significant potential for future improvements.

2. Summary of the Original Paper

2.1 Methodology of the Original Paper

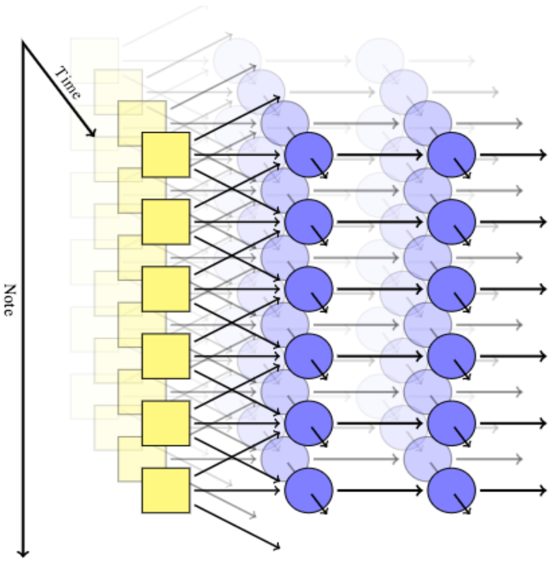
Johnson's paper begins with a general background on neural networks, recurrent neural nets with LSTM in particular, and then delves into the process behind how we may be able to use these to compose music. His design outlined a few key requirements for his RNN implementation created to ultimately compose music:

1. Built-in understanding of time signature
2. Time Invariant
3. Roughly Note Invariant
4. Supports chords / multiple notes
5. Allows for repeated notes (with an explicit distinction from held notes)

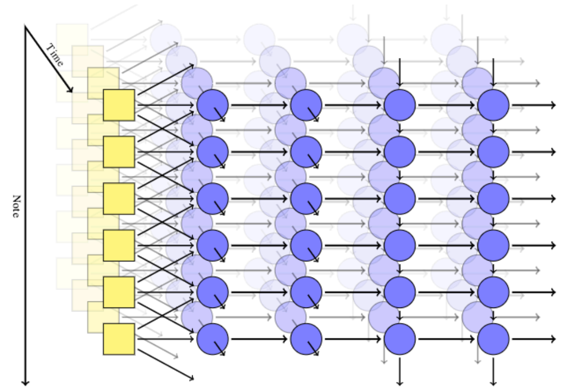
Johnson highlights one particular clarification on #3, note invariance, stating that it is typically not present in other approaches for music composition with RNNs, but is crucial in the understanding of music. The system must have a sense of relativity when looking at certain notes (e.g. A C-major chord sounds more like D-major than C-minor despite closer proximity to C-minor). The network must be able to recognize transposes of these notes, rather than the absolute positions.

A network that is commonly used today that carries a relevant invariance property is a Convolutional Neural Network (CNN). Using kernels in CNNs mimics this transpose property as it is merely applying the same function across nearby pixels in an image. Likewise, we may apply this strategy analogously using notes rather than pixels. Typically these convolutional kernels just represent functions, this brought Johnson to hypothesize that we can we utilize a RNN in place of this kernel to create the structure of the network we are looking for.

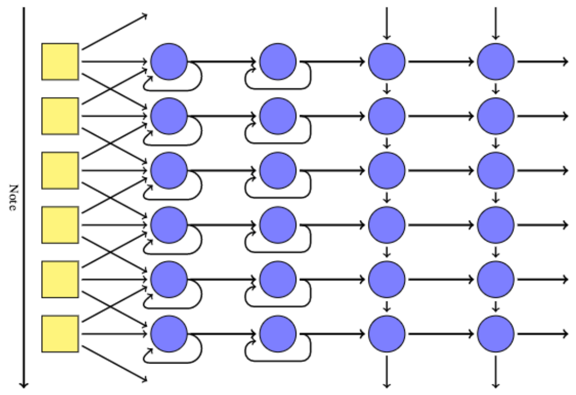
The image below represents a simplified system of inputs that we may now feed into our network.



We have different notes that run along the y-axis, and along the z axis. We can see that we are representing the relative relationships between these notes. Johnson coined the term "Biaxial RNN" to refer to this network structure. The issue with this, however, is that there is still way of identifying chords, due to independence among note outputs. We can fix this by appending additional hidden layers, structured similarly to that shown in the graphic below:



With this implementation, the first half of the network is what allows us to have time invariance with inputs in hidden layers running across time steps. The latter part of the network permits note invariance, including identification and computation of chords due to layer inputs provided from neighboring notes. A simplification of this network is provided below:



We can see that the first half of the network begins to look a lot like traditional LSTM cells in a RNN, which is an interpretation consistent with the eventual implementation.

The next important aspect revolves what the boxes shown in the diagram (inputs) actually are comprised of. Johnson's input vector consists of components detailed in the following table:

Input Vector Component	**Description**	**Purpose**	**Element Count in Input Vector**
Position	MIDI value of current notes	Note invariance and chord support	1
Pitchclass	One-hot encoded vector of all half-step notes	Note Invariance and chord support	12
Previous Vicinity	Relative position (and articulation indicator) of surrounding notes in previous time step, spanning 1 octave in both directions.	Distinctly identifies repeated notes from held notes	50
Previous Context	Count vector for notes played in the previous time step.	Time invariance and distinction of repeated notes from held notes	12
Beat	Position within the measure, under 4/4 time	Interpretation of time signature	4

These inputs are then fed into the network's LSTM stacks that allow us to detect patterns across both time and note axes. These stacks are represented by the hidden layers present in the previous diagrams. Johnson uses 300 LSTM nodes within each of first two sequential hidden layers on the time axis. Johnson then feeds the outputs of the second hidden time-axis layer into the first note-axis hidden layer. There are two total note-axis layers with 100 and 50 nodes respectively (non-LSTM). This input format and network structure yields our desired "biaxial RNN".

The output consists of a "play-probability" vector which identifies whether the likelihood of composing a particular note, paired with an "articulation probability" vector which creates a distinction between held notes and repetition of the same note (satisfying requirement #5).

With full structures for the network and input vector defined, Johnson then selects the raw data he uses to train the network (after pre-processing into proper input vector format). The model outlined was trained on classical music in MIDI format, provided by the Classical Piano MIDI Page [3]. As a step in data pre-processing, Johnson utilized a batch size of 10 randomly chosen 8-measure chunks, selected out of the entire dump/discography available on the Classical Piano MIDI Page.

With network/input structures identified paired with complementary pre-processed data, training begins using cross-entropy as a loss metric and an AdaDelta optimizer. Dropout of 0.5 is used in each hidden layer, indicating batches used in training randomly ignore half the nodes' weights in that layer. Dropout is implemented with the intention of generalization, preventing overfitting to the selected dataset.

The final step revolves around composition. We can use our trained network to "predict" given the outputs of the previous layers. Iteratively stepping by one step on the time axis, followed by a full iteration over the note axis to provide inputs for the following time step allows us to compose music using our network. Due to dropout used in training, one must properly account for the lack of dropout in composition, as it is imperative to utilize all node weights when composing for optimal results.

2.2 Key Results of the Original Paper

Contrarily to many supervised learning problems, it is quite difficult to provide an easily interpretable measure for quality of the composed songs. "Accuracy" isn't quite possible with musical composition like it may be for other tasks. We rely on loss minimization in training our network, but this truthfully doesn't give us an accessibly interpretable metric to measure the success of the implemented RNN solution.

Qualitatively, many of the examples that Johnson provides as examples of his final networks compositions are quite impressive. As results are difficult to visually depict in this paper, we have provided audio samples available in the project's repository in the following location:

*./generated_music_johnson/**

The major concern both Johnson and myself have with the results presented in his research is the excessive holding of certain distinct chords. The network seems to hold chords for extended periods of time that doesn't quite sound natural. Other than this minor detail, we find that heuristically, the pieces are quite passable as ones made by human composers.

3. Methodology

3.1. Objectives and Technical Challenges

The general goal for this project using Recurrent Neural Network is to compose music. Using music as input data, the network will be trained to predict the next most probable to be played, ultimately composing a full piece.

There were a few major technical challenges that monopolized the majority of our efforts:

- LSTMs are quite conceptually complicated and also difficult to implement properly
- Preprocessing for music can be more complex than standard datasets. Domain knowledge becomes particularly useful in a problem like this, and lacking it becomes an obstacle in and of itself.
- The density and structure of the network certainly makes it more difficult to understand. Concepts like stacked LSTM cells and multiple axes for dimensions complicate the model greatly.
- Johnson's code is implemented in Theano, a package that none of us were initially very familiar with.
- Measuring results is complicated. As mentioned prior in (2.2), we don't have a simple accuracy metric like we might have with other supervised learning problems. We must rely on a generic loss in order to interpret our results.

3.2. Problem Formulation and Design

The overall implementation of this problem was divided into three parts: data processing, modeling, and training. These parts are completed sequentially in the following set of modules listed below.

<u>Data Processing Module(s):</u>

midi_to_statematrix: with two core functions called *midi_to_note_statematrix()* and *note_statematrix_to_midi()*. This module is used for conversion between .mid file and statematrix (a list based data structure used to store input data).

data: with core function called *note_statematrix_to_inputForm()*. This function is used to convert statematrix as input to the RNN.

<u>Modeling Module(s):</u>

model: We define *Model()* class in this module. The note wise input size is 80. Then we specified time axis network structure and note axis network structure. Each network is based on stacked LSTM cells with dropout applied.

Training Module(s):

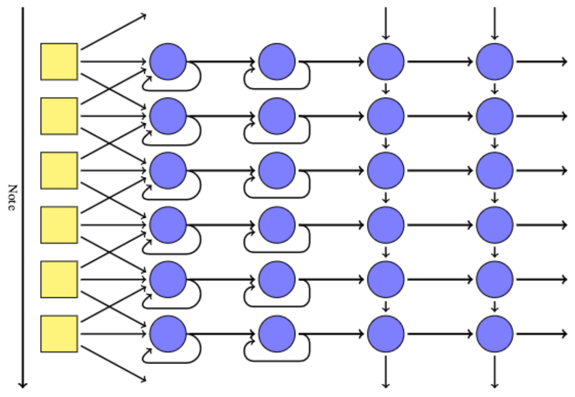
multi_training: with two core functions *load_pieces()* and *train_piece()*; *load_pieces()* is used to read .mid file in the directory and output a dictionary; *train_piece()* is used to train the RNN with several parameters specified: time axis network structure and note axis network structure.

4. Implementation

The implementation we used intentionally mimics that of Johnson's research and blog post. Using his code as a starting point, the main task revolved around converting the code that was traditionally written in Theano and modifying it for Tensorflow.

4.1. Deep Learning Network

As we sought to replicate Johnson's paper, we retain the same architecture that was presented in the section above (2.1). To reiterate, below you may find the general structure of the network:



The architecture of the implemented network contains two hidden LSTM layers to produce invariance of time and note. Before each step, data was reshaped to be compatible with batch sizes. The first two hidden layers' intention is to train with respect to the time axis. It is comprised of a two-layer dynamic RNN model with LSTM cells, whose size is (300, 300).

The latter two hidden layers' purpose is to produce quality sounding music via the usage of chords and relativity amongst notes. These layers are still part of the RNN structure with the first two layers, except we only include (100, 50) nodes respectively and these are not LSTM stacks.

The last layer (not depicted) is a fully-connected layer that outputs a 2-D vector, representing the probability of play and articulation, just as described in section (2.1).

The network is trained by minimizing a sequence of a mean log-loss function with added noise and optimized using an Adam optimizer.

4.2. Software Design

The general design pattern is described in section (3.2). To begin, we first created a "music" folder in the parent directory, and placed all music files into this subfolder (in proper .mid format) to use as our training data. We then import the *multi_training* module to read our MIDI files and pre-process to the proper input format for the neural network.

Next we import our model from model.py ensure that we are using the proper structure outlined by Johnson (it is pre-defined as *biaxial_model* in the Theano implementation). We must ensure to specify the time axis network size and note axis network size.

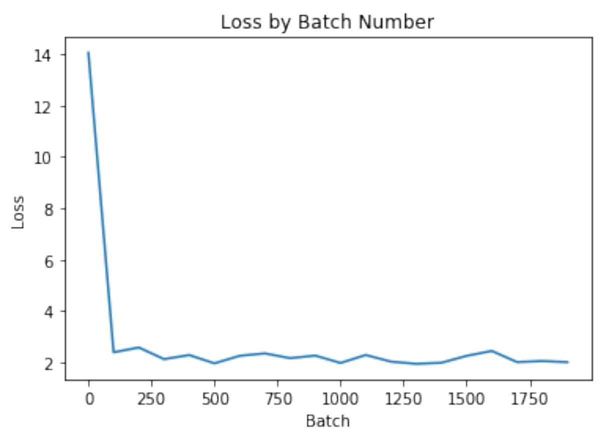
We now have our network and can begin training. We apply the training function (available in multi_training.py) on the model. We utilized a Google Compute Engine instance equipped with a Tesla K80 GPU for training purposes, training for roughly 12 hours. Due to resource constraints, we used a significantly smaller training set than Johnson had mentioned in his research, only 15 songs compared to the full dump of MIDI files. As we are not using "songs" directly as one training example, but rather 8-bar measures within the songs, we felt this may still be appropriate.

We are then able to use our trained model to generate a .mid file to the output folder (make sure this folder exists before run the model). This MIDI file represents the composed music of our neural network and is then immediately available for listening.

5. Results

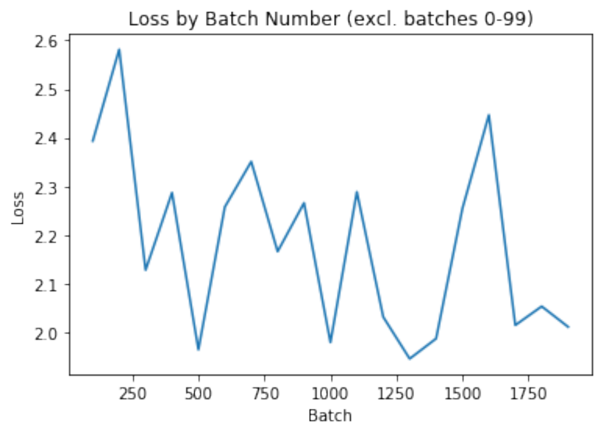
5.1. Project Results and Comparison

As mentioned in section (2.2), results within composition are inherently more difficult to discuss than many traditional machine-learning projects due to a lack of a real "accuracy" measure. That being said, we are still able to view, generally, how training affects the loss of our model in the following figure.



Note: Loss values only reported every 100 batches

We see above that we experience the standard exponential shape to our loss that we typically see in many machine learning problems. We see rapid improvement in the model's performance in early batches of training, and experience extreme diminishing returns as we continue to train further. It is a bit difficult in the graphic above to interpret whether or not we are still improving in later batches. Let's take a look at our loss beginning at batch 100 so as to not obscure the trend with an exceptionally high loss in the beginning of training.



Here we can see that our training loss, while experiencing some variance, is generally still decreasing even at the end of our training session. Time permitting; it seems likely that continuing to train our model would yield further improvements.

As loss alone doesn't intuitively illustrate the quality of the compositions, we have provided audio recordings from three distinct sources and their respective locations within the project repository:

1. Baseline composition from the untrained network:
   *./untrained_baseline/**

2. Compositions produced from Johnson's trained model:
   *./generated_music_johnson/**

3. Compositions produced from our trained model:
   *./generated_music/**

The baseline model's composition seems to lack any form of variation in note/chord choice or tempo. It merely sounds as if someone is repetitively mashing the same piano keys at the same rate.

The compositions produced from Johnson's trained model on the other hand, as mentioned in (2.2), are quite impressive. There is a clear understanding of timing, relative note and chord choice, and sounds natural in terms of music.

The resulting compositions produced from our model, as anticipated, lie in the gray area between the prior two models'. There is a distinct improvement over the untrained, randomly initialized model. We hear variation in note and chord choice, and a stronger sense of timing rather than merely playing the same note or chord at every beat in every measure. That being said, our model falls far short of Johnson's in terms of quality in composition. It lacks a certain elegance that traditionally accompanies music; the notes that are chosen are not particularly complementary to one another. Likewise, we don't quite hear an awareness of audibly pleasing tempo or beat changes like we may hear in Johnson's compositions.

The author of the original paper did not report precisely how long he had trained for. As mentioned prior, we opted to train for around 12 hours using a Tesla K80 GPU, looking at approximately 2,000 batches of 10 eight-measure segments per batch. A very likely explanation for the aforementioned problems in our composition revolves around incomplete training and insufficient data as discussed in the following section.

5.2. Discussion of Insights Gained

We are able to say that a great deal was learned throughout the production of this project.

First and foremost, we learned that problems like musical composition are solvable using RNNs if structured in a creative way. While Johnson proposed a seemingly simple structural outline of the network, the actual implementation involves an intimate knowledge of how LSTMs and RNNs in general function, particularly in practice.

Through some lackluster results, we also experienced a valuable lesson in the training process. Allotting the proper time to train some of these more complex models is sometimes the only missing piece of the puzzle in obtaining meaningful and appropriate results. While our process may have been correct, if we are merely training for a small fraction of the required time, the model is still going to be in its infancy of learning. In the future, in addition to allowing the model a longer duration to train, we would also utilize more training data permitting for better generalizability of the network we implemented.

Going forward, we would also like to test the generalizability traits of the model. One area that we believe to be interesting, but out of scope of the initial project, is applying this network to other genres of music other than classical. We are curious to see if this network is able to adapt well across genres given the appropriate training data.

6. Conclusion

We sought out to produce a pipeline for composing music utilizing deep learning, specifically with neural networks. Through replication of Johnson's research and blog post, we were able to successfully curate a framework to solve this problem of musical composition.

Our results, comparatively to the original paper's results are admittedly underwhelming, but a great deal of this, as mentioned, can be attributed to the relative lack of resources (in both time to train and computational power). Given the aforementioned resources, we are confident that we would be able to produce significantly more impressive results, converging to the same tier of composition presented in Johnson's research.

Overall, despite obvious room for improvement, we are pleased with our ability to replicate such a complex network and obtain results that we can analyze and refine in future endeavors.

7. Acknowledgement

We are immensely grateful for all of the support and assistance provided to us throughout the process of designing this project and presenting this paper.

The primary source of assistance to us throughout this project was the research and blog post compiled by Daniel Johnson in his paper "Composing Music With Recurrent Neural Networks". Using his framework, we were able to replicate his results and translate it into a proper Tensorflow implementation.

Additionally, a significant portion of this project utilized knowledge obtained in ECBME4040, Deep Learning and Neural Networks at Columbia University led by Professor Zoran Kostic.

8. References

[1] https://bitbucket.org/ecbm4040/2017_assignment2_sr3336/src/TEMP.project.bs2996.kc3041.sr3336

[2] D. Johnson, "Composing Music With Recurrent Neural Networks", *Hexadria*, August 02, 2015. http://www.hexahedria.com/2015/08/03/composing-music-with-recurrent-neural-networks/

[3] B. Kreuger, "Classical Piano MIDI Page", 2016. http://www.piano-midi.de/

9. Appendix

9.1 Individual student contributions - table

	bs2996	kc3041	sr3336
Last Name	Sun	Chen	Reddy
Fraction of (useful) total contribution	1/3	1/3	1/3
What I did 1	Replication of original repository	Replication of original repository	Replication of original repository
What I did 2	Methodology (3.1, 3.2), Implementation (4.1, 4.2)	Methodology (3.1, 3.2), Implementation (4.1, 4.2)	Abstract, Intro, Summary of Original Paper (2.1, 2.2), Results (5.1, 5.2), Conclusion, Acknowledgement, References, Appendix
What I did 3	Primary contributor for Tensorflow conversion from Theano	Primary contributor for Tensorflow conversion from Theano	Formatting, cleanup, revision and QA for the report