Project milestone: Generating music with Machine Learning

David Kang
Stanford
dwkang

Jung Youn Kim
Stanford
jyk423

Simen Ringdahl
Stanford
ringdahl

Abstract

Composing music is a very interesting challenge that tests the composer's creative capacity, whether it a human or a computer. Although there have been many arguments on the matter, almost all of music is some regurgitation or alteration of a sonic idea created before. Thus, with enough data and the correct algorithm, machine learning should be able to make music that would sound human. This report outlines various approaches to music composition through Naive Bayes and Neural Network models, and although there were some mixed results by the model, it is evident that musical ideas can be gleaned from these algorithms in hopes of making a new piece of music.

1 Introduction

This report will explore the various ways in which a computer can be taught to generate music. Having learning algorithms as a creative aid will offer great help for those seeking inspiration and innovation in their music writing. This report will be approaching music generation in four ways: one through a simple Naive Bayes algorithm and the others through neural networks, specifically a vanilla neural network, an LSTM RNN, and an encoder-decoder model RNN. For each algorithm, we will utilize different approaches to data organization and music creation. The Naive Bayes and the vanilla neural network will organize the notes temporally, where a song is outlined by the specific time interval in which a note is played as each note will have a start and end time within a given piece. For the LSTM model, a song will be outlined based by new note events, where every time there's a new note, a new vector is created. The model also takes in one note at a time and outputs one note at a time. For the encoder-decoder model, the song will be organized by chords, where a song consists of just a series of chords with varying lengths. Moreover, the encoder-decoder model will take as input a sequence of notes and output a sequence of notes.

1.1 Problem definitions

The input to our algorithm will be a note or a series of notes from a MIDI file. We then use a Neural Network, Recurrent Neural Network, an Encoder and Decoder Recurrent Neural Network, and a Naive Bayes approach to generate a new sequence of notes with the aim of making a *good* piece of music.

To test how successful the music generation is from the neural networks, we will evaluate the prediction of the next note against the actual next note as a percentage score. For the Naive Bayes approach, however, we will compare the generated music to a random (flat) distribution of notes. We will therefore need to ask peoples opinions on to what extend our model improved the quality of the music. This is a slightly unscientific approach of evaluating music, but then again, that is the nature of music.

2 Related work

There have been many attempts to make music using machine learning. One of the first attempts, made use of Markov Chains, and transition matrices that define the probabilities of certain notes being produced. This work was carried out by Iannis Xenakis (20) The early successful works in this field began by looking at sequences, trying to predict the note played at some point $n+1$ using all n points. Peter Todd (17) was among the first at this.

In the past few years, there has been an increased interest in machine-learning-created music. Ranging from AIVA (1) which focuses on making complete soundtracks using AI, to the Magenta research project (11) which touches of many different aspects of music production.

In addition to these rather large scale projects on music, the more successful approaches in music creation have been results of neural networks. There have been smaller projects looking particularly at the MIDI-build up of music. Daniel Johnson managed to make a simple piece of music using a specially designed recurrent neural network (RNN) (7), and RNNs have repeatedly been shown to give good and useful results in music generation. Drawing inspiration from these works, we will also apply RNN-algorithms to model sequences of music.

Moreover, a recent Transformer model has done very well in creating music that most resembles actual human performance. A recent study by Huang et al. (2018) (6) utilized data that took into account the velocity the notes being played along with using models along with using a sequence model with self-attention to best maintain the medium-term memory that exists in music.

There have been far fewer attempts at making music using classification algorithms such as Naive Bayes, and they show variable results. The approach is usually combined with numerous strong assumptions and is thus not necessarily as versatile as the RNNs. (10)

3 Dataset and features

We had two main ways of obtaining data for our processing. The first was to use free downloaded files of MIDI-music and then use music21 (4) to decompose the files. For the encoder-decoder model that we've implemented, we focused on mainly classical piano music from the website www.piano-midi.de/ because of the availability of MIDI

files for classical music and the fact that the music for classical music is more standardized than that of another genre. Piano works from a variety of different composers such as Bach, Brahms, Beethoven, and Mozart were used for a total of 771 songs. We processed all the MIDI files using the music21 Python package (4), and for MATLAB we used a similar written package (15). Each song was converted into a collection of chords, where all the notes were "chordified" or in other words compressed into a single chord for each time step. In other words, at every new note event (i.e. a new note was played), a new chord will be created to result in that change. Moreover, each chord was represented as a multi-hot vector of length 219. 128 for the number of possible of MIDI notes, 64 for the possible pitch duration which was derived from the given data, 8 for the possible numerators of the time signature, 4 for the possible denominators of the time signature, and 15 for the possible key signature (represented as number of flats, positive or negative). Thus, a song might consist of 5000 chords with varying note duration, which is our "note" representation as previously mentioned contrasted to the temporal representation. A total of around 700 songs were used for a total of around 400,000 notes, and a standard 80-10-10 training-validation-test split was used.

For the LSTM model, the dataset consisted of 24 of Romantic composer Frederic Chopin's etudes from https://www.classicalarchives.com/ for a total of 57 minutes and 25 seconds of music and 22,290 notes. Due to the structure of piano music, we decided to process notes (or chords if there were multiple notes) and rhythms of the both the left and right hand at each time stamp to train our model.

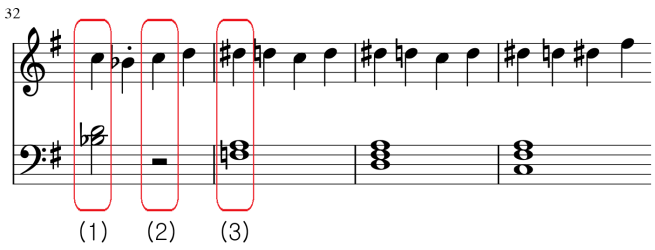

Figure 1: Beethoven's Fifth Symphony music sheet

For example, in fig. 1, the data from (1) would be represented by the vector ['C5', 1.0, 'B-3.D4', 2.0], where the first two entries refer to the notes played by the right hand and their duration while the other two entries refer to that of the left hand. If only one hand is playing at a time as in (2), it would correspond with an entry with a None instance. Thus, (2) would be represented as ['C5', 1.0, None, None]. Thus, a song would consist of a series of these vectors for each note event. Then a dictionary was created to enable one-hot encoding for each of the unique vectors. Therefore, as seen in fig. 2, our data consists of a three dimensional matrix with the height corresponding to each song, width corresponding to the one hot vector of note incident, and the length corresponding to the the time stamp of the note incident.

Aggregating non-classical music for the Naive Bayes and neural network model proved to be difficult for various different reasons ranging from copyright issues to general availability. Thus, we have referred to another method for collecting data.

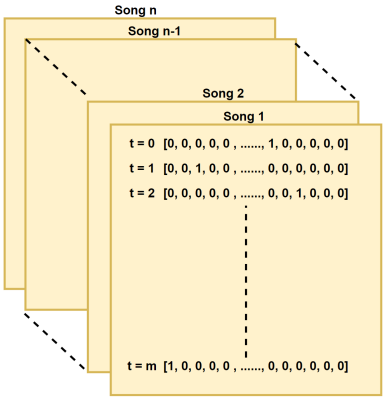

Figure 2: Matrix representation of LSTM Model data

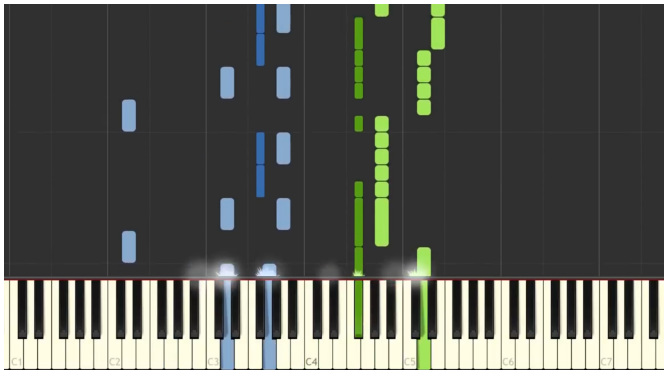

Figure 3: A MIDI-file played in Synthesia

fig. 3 shows a screenshot from a video of a very popular piano software Synthesia (16) playing a MIDI-file as if it were being played on a real piano. The piano keys change color when a particular note is played, and usually, the right and left hand notes are colored differently. These videos can range from single 3 minute songs to hour-long videos with for example *Pirates of the Caribbean* film music. We therefore made a program which downloads and looks through these videos and generates an array with the notes played and at what times and their duration. This proved to be a working way to gather data on much more recent music, which might not have MIDI files readily available to the public. This data will give us the key pressed, expressed as an integer ranging from 1 to 88 representing the number of keys on the piano in order from lowest frequency to highest frequency, and it is organized temporally by the start and end time of when the note was pressed.

4 Methods

As outlined previously, we have utilized four models of Naive Bayes, vanilla neural network, LSTM RNN, and the encoder decoder RNN.

4.1 Naive Bayes-like

We have named this a Naive-Bayes *like* approach as only uses some of the fundamentals from the Naive Bayes we know. In this algorithm we look at each press of a note as an independent variable (even though we know that they are not). The purpose of this model is to make a distribution for which keys are pressed for a given chord. Therefore, this algorithm will consist of two parts:

1. **Classify Chords:** This was achieved by looking at the notes played by the left hand on a piano.

We made a program that makes a dictionary of all the unique combinations of three consecutive notes played by the left hand.[1]

2. **Classify notes:** Now that we know the chords in a song, the program will run through many pieces of music and find which notes were played by the right hand for a given chord in the dictionary. After multiple songs, we have generated a comprehensive distribution of $P(\text{note}|\text{chord})$.

4.2 Neural Networks

We implemented a basic neural network (NN) for pattern recognition in music generation. Using a similar method to predicting the next word typed by a user in a text program, we wanted to predict the next note played in the sequence. Therefore we gave the NN a vector representation of the previous note played, its key signature, the start and duration of the note, as well as the previous 100 notes played; a total of 104 entries. We fed this into a neural network with 1024 neurons and asked it to predict the most likely note played among the 88 different possibilities. As we will see in the next methods, there is no direct memory component with this neural network, but having the previous 100 notes as inputs serves as a proxy to the memory methods in the subsequent methods.

4.3 LSTM

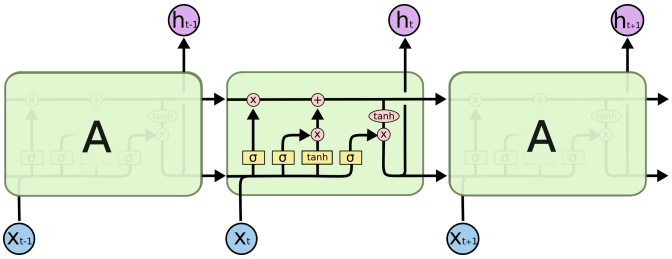

Figure 4: LSTM Structure Diagram (12)

The most popular method for music generation, the LSTM helps us solve the problem of the regular neural network lacking "memory" or knowing how to relate or figure out data sequentially. The main features of the LSTM RNN compared to the regular RNN are the input, output, and forget gates. (5)

$$
\begin{aligned}
i_{(t)} &= \sigma(W_{xi}^T \cdot x_{(t)} + W_{hi}^T \cdot h_{(t-1)} + b_i) \\
f_{(t)} &= \sigma(W_{xf}^T \cdot x_{(t)} + W_{hf}^T \cdot h_{(t-1)} + b_f) \\
o_{(t)} &= \sigma(W_{xo}^T \cdot x_{(t)} + W_{ho}^T \cdot h_{(t-1)} + b_o) \\
g_{(t)} &= \tanh(W_{xg}^T \cdot x_{(t)} + W_{hg}^T \cdot h_{(t-1)} + b_g) \\
c_{(t)} &= f_{(t)} \otimes c_{(t-1)} + i_{(t)} \otimes g_{(t)} \\
y_{(t)} &= h_{(t)} = o_{(t)} \otimes \tanh(c_{(t)})
\end{aligned}
$$

Corresponding to the leftmost part of the center diagram in fig. 4, the first step of the LSTM is to decide what parts of the previous information will be forgotten. This corresponds to the forget gate denoted by $f_{(t)}$. It takes in inputs from the previous hidden state and the current input to combine with the previous cell state to produce the current cell state. For the next step corresponding to the middle part of the diagram, the input gate layer denoted by $i_{(t)}$ combined with the new candidate values $g_{(t)}$ decide what part of the input will be updated into the current cell. To actually update the cell state, the cell state from the previous time step is multiplied by the output of the forget gate and then is added to the input gate. This corresponds to the horizontal line in the top of the center diagram and is noted by $c_{(t)}$ in the equations. Lastly, the actual output and the hidden state for the current time step is denoted by $y_{(t)}$, which is a result of a tanh layer from our cell state and a sigmoid layer from our input and previous hidden state.

4.4 Encoder-Decoder (Seq2seq)

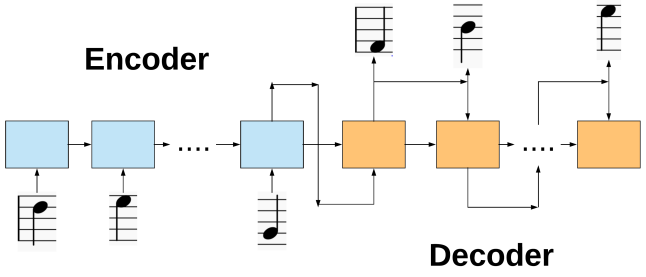

Figure 5: Encoder-Decoder Model Diagram

To further explore the RNN architecture, we have also implemented the encoder-decoder model that is often used in Natural Language Processing (2). While still in essence an RNN, the main difference is that it is a many to many model. In other words, this is a model that takes in a sequence of data and outputs a sequence of data. The encoder takes in the input and "translates" it through the decoder for an output. Moreover, instead of focusing on a very long term memory, the memory of the algorithm is limited to the sequence that we feed into it (100 notes in this case). We have also used GRU layers instead of LSTM layers.

$$
\begin{aligned}
z &= \sigma\left(x_t U^z + s_{t-1} W^z\right) \\
r &= \sigma\left(x_t U^r + s_{t-1} W^r\right) \\
h &= \tanh\left(x_t U^h + (s_{t-1} \circ r) W^h\right) \\
s_t &= (1 - z) \circ h + z \circ s_{t-1}
\end{aligned}
$$

The GRU and LSTM models are similar, except that the GRU has a reset (r) and an update gate (z) instead of the input, output, and forget gates of the LSTM. The forget and input gate are combined into a single update gate while the reset gate determines how to combine previous memory with the new input. In terms of overall structure, they are very similar, but the GRU is a much simpler model than the LSTM and has shown to be quicker to train (12).

Moreover, we used the approach of having each song be a collection of notes. Thus, compared to the Naive Bayes model, the data is not discretized by time, and unlike the LSTM model, the data is not organized by new note events. We will also use more features compared to the LSTM model as the data will include information about key signature and time signature. Although it is likely that the neural networks will be able to learn the ideas of

[1] This may sound like many combinations, but in fact many modern songs can get away with just four different combinations. (19)

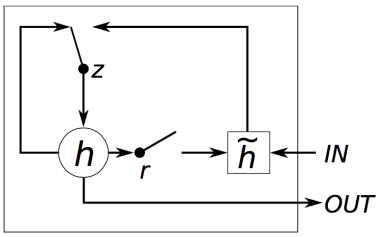

Figure 6: GRU Model Diagram(3)

key and time signature on its own with out the data, we have included it in the feature space to see if there were any different results in the outcome.

5 Experiments/Results/Discussion

Table 1: Results for all the models

Model	Train Loss	Test Loss	Sample Size	Turing Test
Naive Bayes	22.1% (Pred)	3.18% (Pred)	10,570	47%, 93%
Vanilla NN	.01904 (CE)	.1260 (CE)	10,570	47%, 53%
LSTM	.5345 (CE)	.7986 (CE)	22,290	36%, 89%
Enc-Dec	.04473 (BCE)	.06773 (BCE)	408,700	23%, 30%

Sample size is # of notes. Turing Test is % guessed human, % perceived as intermediate/advanced composing level. All models had 15 survey responses

5.1 Vanilla Neural Network

We downloaded a *Pirates of the Caribbean* (9) film music compilation and found it had close to 10,000 notes played. Feeding this through our artificial neural network (hilariously termed *Depplearning* for the occasion), it was trained over 115 iterations. Using 15% of the data for validation and 15% for testing, the neural network showed the following development:

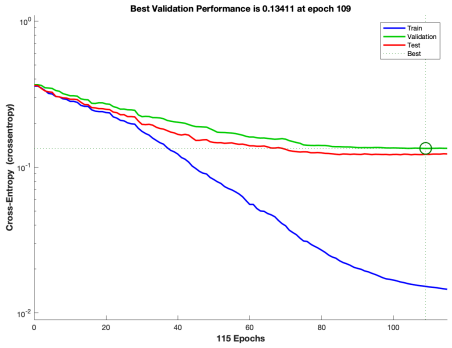

Figure 7: Neural network performance on a dataset with ca. 10,000 values, with 1024 neurons.

The network showed a 5% error on the training set, but a 88% error on the test set, which is a sign of overfitting. When using the network to make music, we heard that the same notes were repeated unnaturally many times, and the song had no weight on harmony. It is as if the network gives up in finding any pattern, and instead returns the most played note in the selection.

The way we would generate the song after having trained the network is to keep the timings (start time and duration) of the original notes in the song, and then only change the note played. This further means that we have completely ignored the fact that more keys are played at the same time, which results in a bigger error for the model. If we would continue this project further, then this would have been an important area of focus.

To qualitatively assess the the music itself, we conducted a music Turing test survey. We let survey participants decide whether or not each generated piece of music by the algorithm sounded human or computer generated. We also asked participants what they thought of the composing complexity of each song. 47% of participants thought that the music was human generated while 53% of participants thought that the composing level was intermediate or advance. This is likely due to the model overfitting on the original song, which resulted in a song that sounds familiar with the original song with a few differences.

5.2 Naive Bayes

To test this model, we chose music with comparatively few utilized chords. *Virginia* which had close to 5,000 notes through the song showed a total of 40 different combinations of left hand progressions, with four of these being very over-represented. These are the four chords that are repeated over the entire song. After making the distribution over the notes, we generated a new song where for each chord in the song the algorithm would pick notes from the probability distribution. To test the accuracy of this model, we compared the predicted *key signature* of the note to the actual key signature, and we could obtain an accuracy of over 20 %. However, the point of this algorithm is to consider the harmony of the generated music. This song sounded "better" in the form that we could hear the harmony a lot clearer, a lot more than we would see in a flat distribution. In our survey results, 47% of participants thought that the music was human generated. 93% of participants thought that the composing level was intermediate or advanced. This is again likely due to the overfitting of the model.

5.3 LSTM

Our LSTM model was implemented on NumPy (13) resembling an implementation from Navjinder Virdee (18) and consists of a single LSTM layer with 256 hidden neurons. The result of this layer is then passed into a softmax layer for the final output. Loss was calculated through cross entropy, and the algorithm was optimized using the Adam algorithm (8) with a .005 learning rate, $\beta_1 = .9$, and $\beta_2 = .999$.

Figure 8: LSTM Model Loss

The training and evaluation loss dropped substantially in a similar pattern in first 20 epochs and finished at

around .5345 and .8297 respectively after 100 epochs. In parallel, the average test loss was approximately .7986. Looking at the resulting sheet music from the algorithm, qualitatively, it is very difficult to determine whether or not the music was generated through an algorithm.

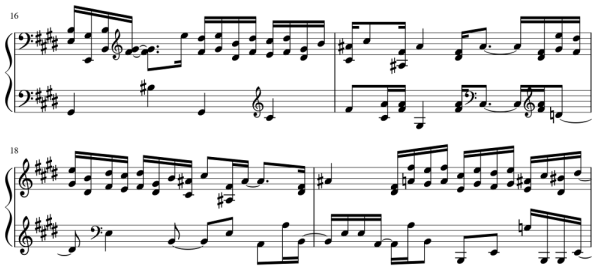

Figure 9: Music Generated from LSTM Algorithm

Although there seems to be repetitions of certain melodies, the notes were slightly offset, so they are not exactly similar. With regards to the survey results, the LSTM model had the most interesting results. Only 36% of participants thought that the music was human generated, but 89% thought that the composing level was intermediate or advanced. Also, most people commented that they were able to distinguish the generated music because it was lack of dynamics. From what we gather, this indicates that the people thought that the music was relatively complex and coherent with typical musical motifs, harmonies, and melodies, but the music lacked a certain human quality. It would be interesting to study in future work what qualities in music lead people to think it to be more human.

5.4 Encoder Decoder

The encoder-decoder model was implemented on Pytorch (14). Batch size of around 100 chords was used, which is around 40 measures or so in a typical song. The Encoder and Decoder both consist of two GRUs with 512 hidden neurons each followed by a dropout layer with .5 drop rate to prevent overfitting. However, the decoder model uses teacher forcing with rate .5 to aid and speed up training. The output of the last dropout layer in the decoder is then passed through a linear and a sigmoid layer that outputs the probability of each the occurrence of each pitch value, note duration, time and key signature.

Loss was calculated through binary cross entropy loss, where both the beginning of sequence and end of sequence tokens were removed to have sensible calculations. The algorithm was optimized by the Adam algorithm (8) with a learning rate of $1e-5$. The low learning rate and gradient clipping was utilized because we experienced exploding gradients in testing our algorithm.

The average training loss went down significantly throughout the epochs and lingered around .04473, but the average evaluation loss plateaued rather quickly and slowly started to increase in later epochs, hovering around .7212. This was closely aligned to the average test loss at .6773. The initial dip in the evaluation loss shows that the algorithm did learn a little bit, but the plateau indicates that there was a rather quick peak in how much the algorithm could learn, and running more epochs would only result in over-fitting. The middling evaluation loss is evident in the music, which consists of very repetitive sounding notes. Music was created by feeding sequences

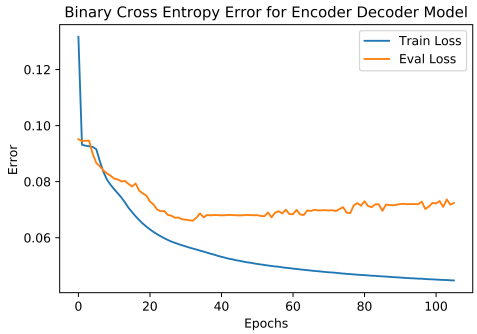

Figure 10: Encoder Decoder Model Average Loss

of notes from the test set. Although there are hints of harmonic progression in the music, it is very clear that the music is not human generated. In our survey results, only 23% of the participants thought that the music was human generated and only 30% thought that the composing level of the music was intermediate or advanced.

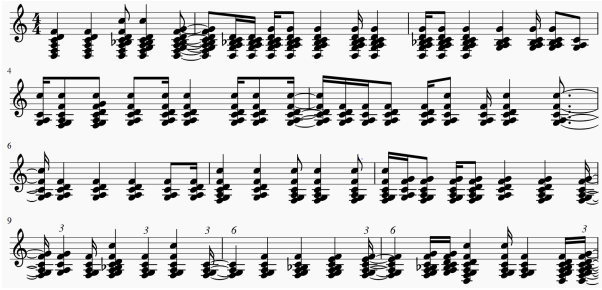

Figure 11: Encoder Decoder Model Generated Music

There are two possible causes for such mediocre performance on songs generated by the encoder decoder model. The first is data structure. Because a single note change indicates a new chord, often times, the notes generally seem as if they are repeating with minute differences each time, which is reflective in the generated music. The second is the small batch size. While 100 notes seem like a very sizable amount (it is around 40 or so measures), for high tempo frenetic songs with many minute changes, there might be high variability in how long these sequences are temporally. Thus, a thing to try in future attempts is to change the batch size for longer term memory.

6 Conclusion/Future Work

In conclusion, among the four algorithms that we tested, the LSTM RNN model performed the best. Coupled with organizing the data through new note incidents, the LSTM model was able to string together musical motifs with coherent melody and harmony. Although Naive Bayes and the Neural Network produced very musical ideas in song, it was largely due to over-fitting the data. For the encoder model, due to the discretization of the data, most of its results were very repetitive notes that made some harmonic sense but was mostly unmusical. In the future, with more time, it would be interesting to explore more memory focused models such as the transformer model while also incorporating note velocity from human recorded MIDI files to bring more life into the generated music.

7 Contributions

With regards to the models, Simen worked on the Naive Bayes and the vanilla neural network; Jay worked on the LSTM RNN; and David worked on the encoder-decoder model. Simen did majority of the work in constructing the poster and writing up the outline and milestone while David helped synthesize the write up/final report. Jay conducted the survey on generated music and accumulated the results.

8 Code and Music

All code can be found in https://www.dropbox.com/sh/ttzb502hheo9fst/AACx3uQSE_CSxnyp6bvaMQzZa?dl=0, and you can listen to our generated music in these links: https://docs.google.com/forms/d/1W3rfI_kevqqDZGk1CnvoGCrLmhzajHHE6hcOfKrKuXw/viewform, https://bit.ly/2zZSGgD.

References

Aiva. (n.d.). https://www.aiva.ai/. Retrieved from https://www.aiva.ai/

Cho, K., van Merrienboer, B., Gülçehre, Ç., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, *abs/1406.1078*. Retrieved from http://arxiv.org/abs/1406.1078

Chung, J., Gülçehre, Ç., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, *abs/1412.3555*. Retrieved from http://arxiv.org/abs/1412.3555

Cuthbert, M. S., & Ariza, C. (n.d.). *music21: A toolkit for computer-aided musicology and symbolic music data.*

Géron, A. (2017). *Hands-on machine learning with scikit-learn and tensorflow: Concepts, tools, and techniques to build intelligent systems.* O'Reilly Media. Retrieved from https://books.google.com/books?id=bRpYDgAAQBAJ

Huang, C.-Z. A., Vaswani, A., Uszkoreit, J., Shazeer, N., Simon, I., Hawthorne, C., ... Eck, D. (2018, September). Music Transformer. *arXiv e-prints*, arXiv:1809.04281.

Johnston, D. (2015, Aug). *Composing music with recurrent neural networks.* Daniel Johnson. Retrieved from http://www.hexahedria.com/2015/08/03/composing-music-with-recurrent-neural-networks/

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, *abs/1412.6980*. Retrieved from http://arxiv.org/abs/1412.6980

Landry, K. (2016, Jun). *Pirates of the caribbean medley [piano tutorial] (synthesia) // kyle landry sheets/midi.* YouTube. Retrieved from https://www.youtube.com/watch?v=iddFrfC0YoU

Lichtenwalter, R., Lichtenwalter, K., & Chawla, N. (2009, 01). Applying learning algorithms to music generation. In (p. 483-502).

Magenta. (n.d.). https://magenta.tensorflow.org/. Retrieved from https://magenta.tensorflow.org/

Olah, C. (n.d.). *Understanding lstm networks.* Retrieved from http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Oliphant, T. (2006). *NumPy: A guide to NumPy.* USA: Trelgol Publishing. Retrieved from http://www.numpy.org/

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., ... Lerer, A. (2017). Automatic differentiation in pytorch. In *Nips-w.*

Schutte, K. (n.d.). *Kenschutte.com.* Retrieved from http://kenschutte.com/midi

Synthesia. (n.d.). *Synthesia, piano for everyone.* http://synthesiagame.com/. Retrieved from http://synthesiagame.com/

Todd, P., & Loy, D. (1991). *Music and connectionism.* MIT Press. Retrieved from https://books.google.com/books?id=NxycaQH6PeoC

Virdee, N. (n.d.). Retrieved from https://www.kaggle.com/navjindervirdee/lstm-neural-network-from-scratch/notebook

White, A. (2014, Apr). *73 songs you can play with the same four chords.* BuzzFeed. Retrieved from https://www.buzzfeed.com/alanwhite/73-songs-you-can-play-with-the-same-four-chords

Xenakis, I. (1992). *Formalized music : thought and mathematics in composition / iannis xenakis* (Rev. ed. ed.) [Book]. Pendragon Press Stuyvesant, NY.