Analysis of a Biaxial-RNN Music Composition Model on Various Genres of Music

Joseph Cammarata Carol Juneau Michael Norton

Abstract

In this paper we discuss our results when training a Biaxial Recurrent Neural Network designed for classical music generation on new data sets to expand on the capabilities of the model. This is done by finding MIDI data from distinct genres and curating the data set to only contain a single instrument MIDI. This technique is used so that when re-training, the model will find patterns in the melody allowing it to replicate other genres. Initial results are promising, with our re-trained samples consistently resulting in improved results and one genre being identified correctly over 85% of the time. However, the testing procedure only accounted for a small portion of our results and a more complete evaluation is required.

1. Introduction

Machine learning approaches are showing significant progress in replicating and generating music across a wide variety of neural network architectures. This is partially because most music has very structured patterns and rhythms throughout the composition. Different genres of music have different patterns which can be learned which is why we felt that we could expand the reach of an existing model to different genres. We tried to find a model that reduced the complexity of the audio inputs so that we could create large enough data sets to train on and so we could complete the training within the semester. There were various approaches proposed such as using existing models that had spectrogram data to do the music generation. While these models seemed to result in better quality generated compositions, they took very long to train. Additionally, the audio input file type needed to be something we could easily create full data sets for using online resources.

In this paper, our goal was to train a neural network designed to compose music from one genre, on multiple genres and see how well the model generalized. We selected a Biaxial Recurrent Neural Network made in 2015 by Daniel Johnson designed solely to compose classical music. The model weights reported in his paper came from training on a data set from the Classical Piano MIDI Page, which contained a wide variety of classical compositions (Krueger, 2018). MIDI data was used to reduce the complexity of the melody and emphasize the patterned and rhythmic elements of the compositions. We curated 4 separate data sets of MIDI files representing country, jazz, holiday, and video game genres. After re-training and generating compositions on the new data sets, the generated samples were sent in a survey where users filled a multiple choice response on which of the five genres they felt each clip represented. Three of the four additional genres had responses where the correct genre was the most selected answer.

We filtered our MIDI data sets for the three most successful genres so that only single instrument MIDI tracks were included but did not use this technique on the country data set, which resulted in significantly worse performance. This filtering and its error is explained more in Section 3. By analyzing the results there is evidence that the model can effectively generalize more genres than classical. By looking at similarities between the genres tested, conclusions can be drawn and tested in future research about which genres lend themselves to machine learning approaches to composition and what elements of music genres may impact the success of such models.

2. Related Works

Our project is based on Daniel Johnson's model (Johnson, 2017). His Biaxial Recurrent Neural Network was created to generate classical music. The model trains on MIDI data using a recurrent neural network. Each output note has its own recurrent network that learns from the notes around it and the previous time step. The model can then take more MIDI data as input to generate a MIDI file of music based on those inputs and using the learned training weights. Our experiment extends Johnson's model to new genres to evaluate if the same model generalizes to those genres.

One of the models before Johnson's was Doug Eck's LSTM recurrent neural network (Eck & Schmidhuber, 2002). The Long Short-term Memory (LSTM) network keeps error constant through time by preventing changes from unused inputs. The model was created to learn repeated melodies and chord structures, and it was tested using twelve-bar blues. Not only was the model only trained in one genre (bebop jazz), but it also only plays over one chord structure. The model was considered successful for bebop improvisation, but it is not clear if this model generalizes to music with more complex structures.

MIDINet is another music generation model (Yang et al., 2017). It uses a general adversarial network combined with convolutional neural networks to generate music from MIDI data. It was trained on pop music. This model was also only tested on one genre. It is said to work for all genres, but we do not have access to the model and are unable to test it ourselves.

Nicolas Boulanger-Lewandowski created a model to transcribe music with multiple parts (Boulanger-Lewandowski et al., 2012). It also uses a symbolic representation of music, like MIDI or sheet music. This model combines recurrent neural networks and restricted Boltzman machines to predict the notes used in a given piece of music. This model is not used to generate new music, but to write down music the model hears. It is relatively successful at guessing the correct notes played in a piece. The model can learn harmony and melody, but it cannot learn long-term structure like repeated sections. It also cannot learn musical meter, which is related to the timing of the notes.

3. Data

We curated new data sets for five genres: classical, country, holiday, jazz, and video game. This data came from a variety of sources on the internet. We used existing MIDI files because using online MIDI converters produced unreliable results. Converting traditional audio formats, like WAV or MP3, to MIDI can be difficult because the timing of notes or tuning of instruments can be imprecise, creating a MIDI file that no longer sounds like the original. Learning from the correct timing and note values is important to the training of our model. In many instances, we removed songs including accompanying instruments or percussion because the model compresses the data into one MIDI track, which would confuse the instruments and alter the input. The remaining data came from solo piano, which could be fed into the generator without issue.

3.1. Selected Genres

For our classical data set we just used the same MIDI data that Johnson's model was trained on. For our Country data set, there was no filtering done on the data because it was too large to individually listen to each track. Instead we used this data to see if having a large amount of training data would compensate for some of the tracks having data with multiple instruments.

The video game music training data was comprised of soundtracks from popular titles such as Pokemon, Kirby, Legend of Zelda, and Mega Man. These tracks come from older games where there is a distinct synth sound throughout the MIDI samples whereas, with modern video games, there is much more diversity in the soundtracks. Additionally, there were large stylistic differences in the samples based on what part of the story they were from. Boss battles had very uptempo music while tracks from dark, slower levels had a dramatically different sound. This variability is a factor in making the training data hard to generalize and could influence the video game genres success on the survey.

The holiday data was the smallest data set but also came from tutorials on how to play various holiday songs on a piano. For this reason there was very little filtering needed and the composition was already made for piano, making it more similar to the data the model was originally trained on.

Lastly, the jazz training data was collected from various MIDI piano arrangements but needed to be filtered so that there were only single instruments.

4. Methods

We use a music composition machine learning model developed by Daniel Johnson called a Biaxial-RNN; the source code is available on Github. We re-trained this model on music from the genres listed in Section 3, generated new pieces, and compared those to pieces generated using Johnson's original training weights.

4.1. Model Structure

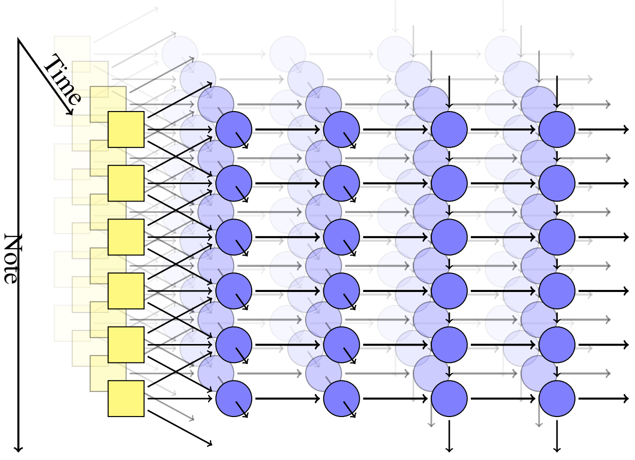


Figure 1. Illustration of a Biaxial Recurrent Neural Network, by Johnson

Johnson calls the model a *Biaxial-RNN* because it has hidden layers that are recurrent along two axes: the time-axis and the note-axis (Johnson, 2015). A "note" is one out of

80 possible MIDI note values, and "time" is the time-step the note is being played in the piece. In Figure 1, see that the direction of computation is along the horizontal axis, notes are along the vertical axis, and time is along the depth axis—coming out of the page (and observe that the figure is a *simplified* illustration of the model). The input and output layers each have 80 nodes, one for each possible MIDI note. These input nodes are shown as yellow boxes and the output nodes are not shown. The input to a node is a vector containing information about the note at the current time-step, including pitch, beat, and surrounding notes.

There are four hidden, recurrent layers, and the nodes in these layers are LSTM nodes, meaning they carry memory across multiple note- or time-steps. The first two hidden layers each have 300 nodes and are recurrent along the time-axis. The last two have 100 and 50 nodes respectively and are recurrent along the note-axis. Finally, the output layer has 80 nodes, one for each note, and each outputs two values: the probability that this note will be played at the next time step, and if this note is the same as the current note being played, the probability that it will be re-articulated.

4.2. Model Implementation

This model is implemented in Theano, a now-outdated library which calculates gradients and compiles GPU-optimized code. The input MIDI files are preprocessed to exclude any that are not in a 4/4 time signature. The remaining pieces are split up into batches of randomly-selected short segments, and these segments are used as input to the model. During training, the AdaDelta method is used to adjust the weights and optimize the cost function (Zeiler, 2012). For the set of parameters x at time t, the formula to find the update Δx_t is given by:

$$\Delta x_t = -\frac{\text{RMS}\,[\Delta x]_{t-1}}{\text{RMS}\,[g]_t} g_t \tag{1}$$

where g_t is the gradient at the t-th iteration. This calculation is done in the code of the Theano-LSTM Python library (Raiman, 2014):

```
dparam = -T.sqrt((xsum + eps) /
    (updates[gsum] + eps)) * gparam
```

which is called for each epoch of training.

4.3. Training the Model

The model utilizes a supervised learning approach during training. That is, for a particular segment used as input, the notes at each time-step are assumed to be the correct output from the notes at the previous time-step. This is how it learns patterns from the training data. Because we already know the *correct* outputs during training, it is possible to optimize the training process by batching all the notes at a single time step together, creating large matrices which the GPU is optimized to handle.

It is important to point out that the model uses a dropout of 0.5 to reduce overfitting. This means that half the hidden nodes are randomly removed (or zeroed) at each layer so that the model does not simply replicate whole sections of a piece (Srivastava et al., 2014). Rather, over time it learns more general patterns from the input data.

However, training this model is a resource- and time-intensive process that took us around 7.5 hours on average using a GPU through the HPC cluster provided by the Research Computing group at the University of South Carolina. Without a GPU, the training process would likely take much longer than would be practical.

We independently re-trained the model on the five different genres three times each, for a total of 15 times, resulting in 15 sets of training weights in addition to the one original set provided by Johnson. All of these weights (which are in a .p file format), as well as the output logs from our scripts, can be found in our Github project repository (see Section 7).

4.4. Generating with the Model

The model is slightly less efficient with composing than with training, because it cannot batch all the notes at one time step at the same time. However, this difference is not reflected when generating a small number of short compositions. Using just a CPU, it was possible for us to generate fifty 15-second compositions in around 140 seconds.

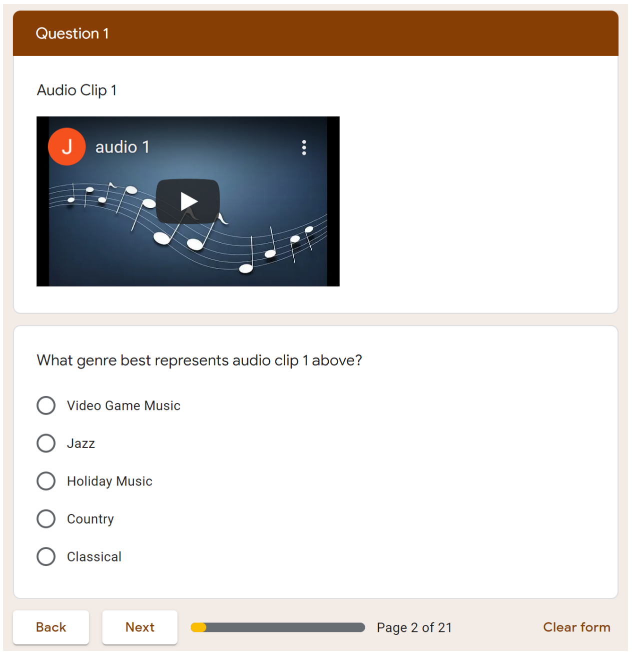
For each of the five genres, we generated 150 compositions; 50 for each of the three independent trainings of the genre. Since the model also uses MIDI files as input for generation, we used the same genres of MIDI files for generation as we did for training. We additionally generated 50 compositions for each genre using Johnson's original training weights for evaluation and comparison.

5. Experiment and Results

5.1. Evaluation Methods

In attempting to evaluate our results we tried a model based and sample based approach. For the model based approach we tried to find a genre classification model that would return the genre that it predicted the sample was and a confidence score. We would feed the generated samples to the model and then take the average of the confidence scores (giving a 0 if the genre was incorrect). The model we chose initially was able to correctly predict many genres; however it did not read in MIDI data, but instead audio data in WAV format. Before we attempted this approach we tried using samples from the training set to validate our approach. The model was unable to correctly get the genres for any of the samples because the data was all piano music and classifier returned jazz for all genres. This illustrated the difference in composition and generation. Our model composed MIDI data, so the differences in the genres were from the structure of the melody, not the instruments used. More complex models that use audio data need to analyze the instruments and other aspects of the sound. This made the model-based approach unsuccessful, and we were not able to implement it in our evaluation. However, in future research, if a MIDI genre classification model is found, or if the generative model also uses audio data in the WAV file type, this approach could be effectively used.

For our sample-based approach, we generated samples for all five genres using the original training weights of the classical model. The model was trained on the four other genres and 150 15-second samples were generated for each genre. Using Google's random number generator, 20 random numbers between 1 and 150 were generated and each respective sample was selected (two from each genre). Using Google Forms, the audio clips were put in a quiz where users were to select which of the five genres they best felt represented what they heard. An example of the question format is shown in Figure 2 below.



Question 1

Audio Clip 1

What genre best represents audio clip 1 above?

- Video Game Music
- Jazz
- Holiday Music
- Country
- Classical

Figure 2. Screenshot from a question in the survey

A link to the survey was sent to friends of the authors of this paper. No metadata was collected about the participants because we did not deem it relevant to the contents of the survey. But, it can be assumed that the population surveyed was mostly made up of college students.

5.2. Bias in Evaluation

Only selecting two clips from the 150 generated per genre results in a very small representation of the generated data set. Ideally clips would be randomly selected from a bank of all the generated outputs. For the ease of getting results, we used Google Forms, which did not support this feature. Additionally, by not including an option for users to indicate that there is no genre they feel the clip accurately represents, some responses may be arbitrary. We did not include this option because if it was available, it may be overused and result in little data about the generated samples.

The survey method emulates the confidence based model method which could evaluate if a sample was more like one genre than others, not just if it directly fit into a genre or not. By keeping out the 'no genre' option from the answers, we force a selection so the results can indicate which genres were closest to the audio sample. Including a genre such as jazz that is so abstract adds more uncertainty to the results. Music genres that are defined by intent and not content cannot be easily evaluated by this method because any composition generated could be seen as a jazz. In future works, a more specific sub genre of jazz such as ragtime would be easier to evaluate with this method.

5.3. Results

After three days, the form was closed with 66 responses. The results from the survey are shown in Figure 3 below, comparing the percentages of correct responses for each genre on the two models.

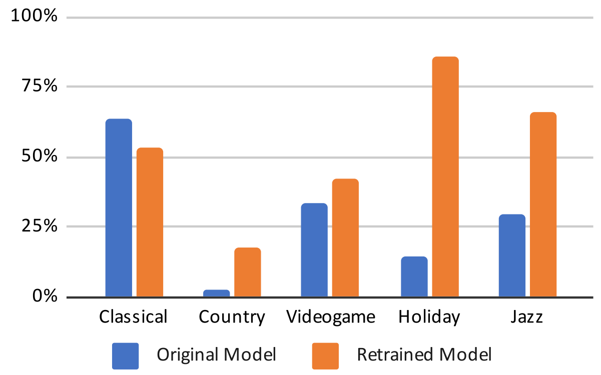


Figure 3. Percent genre was guessed correctly for each genre using original and retrained weights

Additionally, Figure 4 shows the training error for each genre on the retrained models.

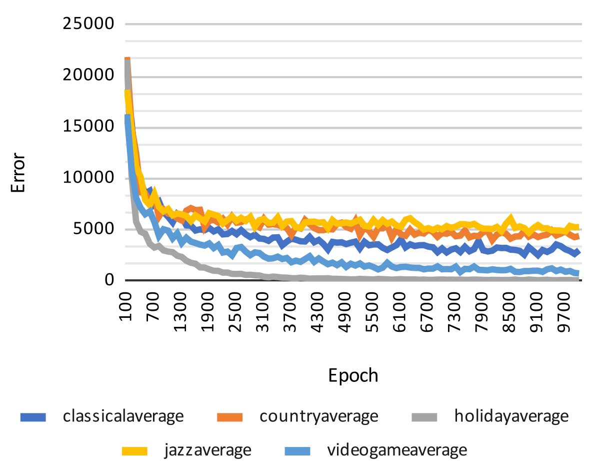


Figure 4. Error when training the model on different genres

The original model only outperforms the retrained models in the classical genre, which is to be expected. However, the genres of jazz and holiday on the retrained model outperform even the original classical model. When comparing the results to Figure 4, it is clear that the lowest error resulted in the best results from the holiday genre. However, the genre with the second lowest error (video game) performed fourth worst out of the retrained models. The jazz and country genres had notably higher error during training. The country set reflects this with distinctly poor results in the survey, but the jazz set ends up having the second best results across all model weights. This may be due to the bias of an abstract genre mentioned in Section 5.2. The lowest errors do not fully correlate with the best results making the success of our evaluation method inconclusive.

5.4. Holiday Genre

While the holiday genre using the original model had a 14.7% success rate (Figure 5), the holiday genre after retraining had over an 85% success rate (Figure 6), the highest percentage of any of the genres. This correlates with the low error seen in Figure 4. The success of the holiday genre could come from the fact that it is the smallest data set. This means that there could be less diversity amongst the generated samples, but all of the samples sound the most like the training data. It also could indicate that the distinguishing feature in holiday music is based on the tempo and melody. If this is the case, it would be easier for the model to replicate the patterns in the music. Holiday music is also highly structured like classical music, so there could be bias towards the holiday training data because the model is designed to be trained on structured compositions.

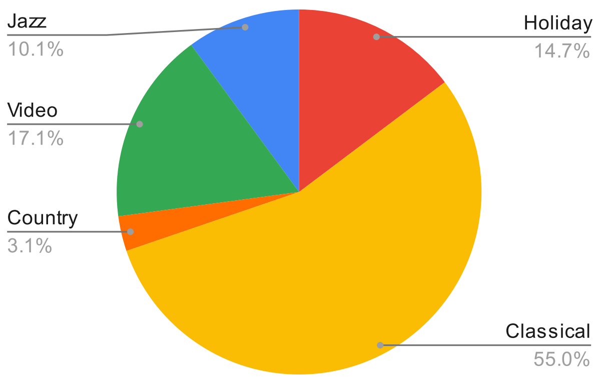


Figure 5. Average results for Holiday samples using original model

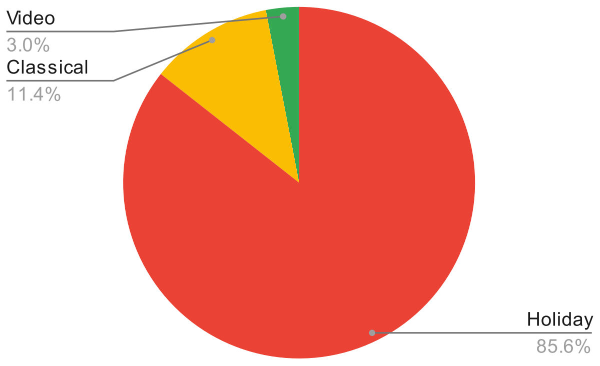


Figure 6. Average results for Holiday samples using retrained model

It is likely that all of these factors addressed contribute to the holiday genres success to some extent. Further research would need to be done to determine the significance of these possible explanations. However, this does indicate that by analyzing other genres on composition based models, a genres dependency on melody and tempo can be assessed. It also suggests that if a model is trained multiple times on a set of different genres, the relationship of the genres within the set can be explored by analyzing the generated outputs of the model.

5.5. Country Genre

There was significant failure in composing music for the country data set. This could indicate that the genre was defined more by the instruments used than the composition of the melodies. This is also the genre that had the least amount of data filtering on the data set. Multiple tracks contain multiple instruments, making it difficult to pick up

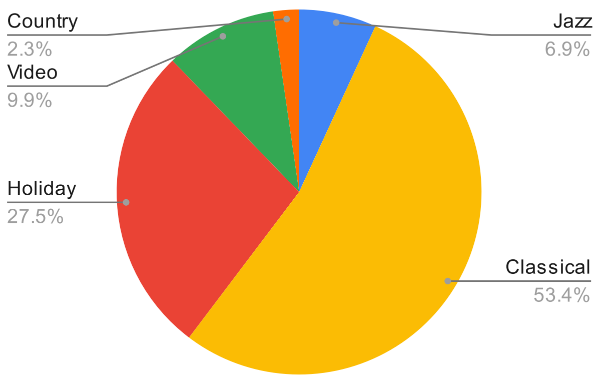


Figure 7. Average results for Country samples using original model

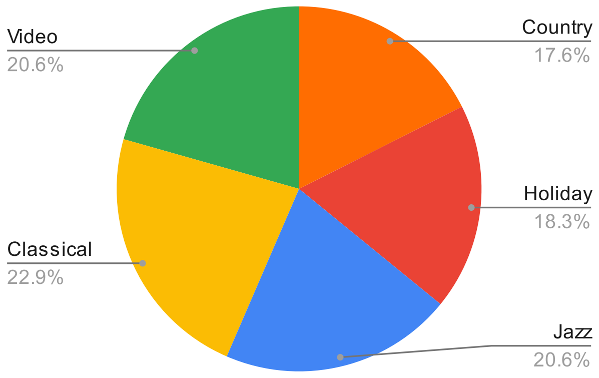


Figure 8. Average results for Country samples using retrained model

on any patterns in the melody.

In future research, a more complete training set with single instrument MIDI data is necessary to determine if the failure of the country set is due to the data set or if there are properties of the genre that limit its success with our model. Figures 7 and 8 suggest that the model can pick some patterns as it performs significantly better than the original model. However, in both models it remains the least picked genre for its samples.

6. Conclusion

The results of our experiment show that it is possible to retrain a Biaxial-RNN model built to compose classical music to also compose music of other genres. Significant improvements can be made when using input data of good quality. However, more work needs to be done to improve the scope and capabilities of this model.

6.1. Lessons Learned

Throughout this project we have learned a great deal about using a particular machine learning system.

6.1.1. Software Difficulties

There were several difficulties in using the software required to construct and run the model. Johnson's model uses Theano as a backend, which is a Python library that is no longer maintained. His code, written in Python 2, is from 2015 and uses some other older Python libraries that have not been recently updated. We had some issues with getting Theano to work with the GPU we had access to, and while in the end we were able to use it successfully, it took a significant amount of time (weeks) just to get the model to work. This limited us in the scope of our experimentation.

6.1.2. Types of Data

MIDI data is ideal to use for this type of model because it contains all the necessary data about a piece with no noise and with a compact file size. However, there are some limitations. It can be difficult to find good MIDI data to use. Some MIDI data on the internet has been converted from an audio file, which would then include some noise. Some MIDI files have multiple tracks with multiple instruments including percussion. Our model does not know how to handle this, as it is built for a single "piano" track. It can play multiple notes (chords) at the same time but does not have multiple instruments, so we need to exclude data that has multiple instruments.

6.1.3. Implementation

Because of the software limitations of Johnson's project, it would be beneficial to develop a new model or adapt the model to a newer backend that is kept up to date. It would also be beneficial to have access to more powerful GPU hardware in order to reduce the training time.

6.2. Future Directions

This project is a good starting point for machine-learning based MIDI music composition. If the model were to be put into production, it would require more fine-tuning of the model structure. For example, one could adjust the model itself by increasing the number of layers or nodes to see if it makes a noticeable improvement. Perhaps one could reduce the number of epochs in training to reduce the training time. If we had more time, we might test out new versions of the model and see how they compare.

7. Supplementary Materials

See our presentation video at: https://www.youtube.com/watch?v=lCkIf_xvrh4

The following supplementary materials are available on the project Github repository at: https://github.com/caroljuneau/biaxial-rnn-music-composition

- Source code
- Code tutorial in the README
- Input MIDI data
- Output MIDI data (our compositions)
- Scripts used for training and generating
- Output logs from scripts
- Metadata spreadsheet from logs
- Survey results spreadsheet
- Link to audio samples used in survey
- Presentation slides

References

Boulanger-Lewandowski, N., Bengio, Y., and Vincent, P. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. Technical report, Université de Montréal, 2012.

Eck, D. and Schmidhuber, J. A first look at music composition using LSTM recurrent neural networks. Technical report, Instituto Dalle Molle di studi sull' intelligenza artificiale, 2002.

Johnson, D. Composing music with recurrent neural networks [online], 2015. https://www.danieldjohnson.com/2015/08/03/composing-music-with-recurrent-neural-networks. Accessed: 12.11.2021.

Johnson, D. Generating polyphonic music using tied parallel networks. In *Computational Intelligence in Music, Sound, Art and Design*, pp. 128–143, 2017.

Krueger, B. Classical piano midi page [online], 2018. http://www.piano-midi.de. Accessed: 12.11.2021.

Raiman, J. Small Theano LSTM recurrent network module [online], 2014. https://github.com/JonathanRaiman/theano_lstm. Accessed: 12.11.2021.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Yang, L.-C., Chou, S.-Y., and Yang, Y.-H. MidiNet: A convolutional generative adversarial network for symbolic-domain music generation. In *Proceedings of the 18th International Society for Music Information Retrieval Conference*, pp. 324–331, 2017.

Zeiler, M. D. ADADELTA: An adaptive learning rate method. *ArXiv*, abs/1212.5701, 2012.